

Aprende Streamlit

Por: Nick Antonaccio (nick@com-pute.com)

Este tutorial demuestra cómo usar la biblioteca Python Streamlit para crear más de 20 aplicaciones web básicas de CRUD y bases de datos.

Contenido:

	1. Cómo ver este tutorial mejor en su dispositivo
	2. Requisitos
previos	3. ¿Qué es Streamlit?
	4. ¿Por qué usar Streamlit?
	5. Primeros pasos
	5.1 Hello World
	5.2 Uso de la biblioteca estándar de Python con Streamlit
	5.3 Interacción con las API
web	5.4 Ejemplo
de una API de Pokémon	5.5 Ejemplo
de	API de base de datos de cócteles
Streamlit	5.6 Más widgets de entrada y salida de
	5.7 Simplemente asigne y pase variables
	6. Flujo de programa Streamlit
	6.1 Diseño
	6.2 Valores persistentes y almacenamiento en
caché	7. Construyendo aplicaciones básicas de base de datos CRUD con Streamlit
	7.1 Un manual
rápido de Python SQL	7.2 (Des)Serialización de listas de Python y otras estructuras de datos en SQLite
	7.3 Conectando declaraciones SQL con widgets de interfaz de
usuario Streamlit	7.4 Terminando una aplicación CRUD básica: Agregando botones para actualizar y eliminar
	7.5 Trabajando con imágenes, sonidos y videos - Image Database App
	7.6 Trabajar con fechas y horas - Aplicación de base de datos de calendario
	8. Algunos patrones de diseño CRUD más típicos, columnas y formularios
individuales	8.1 Ir más allá de los diseños repetidos con St.expander(), usando St.columns()
	8.2 Diseños tradicionales de
un solo formulario	8.3 Usar cuadrículas de marcos de datos para mostrar el contenido de la base de
datos	9. Aplicaciones de ejemplo más simples para demostrar los conceptos básicos de
	Streamlit en la práctica
	9.1 Tiempo entre fechas
	9.2 Visor
de cámara web en vivo	9.3 Coin Flip
	9.4 Recuento
de palabras	9.5 Generador
latino de cerdo	9.6 Calculadora
	9.7 Mad Libs
generado dinámicamente	9.8 Editor
	9.9 Sala de chat (con datos de mensajes guardados en un archivo de texto)
	9.10 Sala de chat (con datos de mensajes guardados en la base de datos SQLite)
	9.11 Chat de múltiples temas (varias salas)
	9.12 Caja registradora
mínima	9.13 Caja registradora Segunda parte: Guardar recibos y generar informes
de ventas	9.14 Reloj
de turno	9.15 Administrador
de archivos FTP	10. Instalación y ejecución de Streamlit en un ordenador
local	11. Streamlit Sites
de varias páginas	11.1 Uso de contenedores y St.Sidebar()
	11.2 Verdaderas aplicaciones de varias páginas desde múltiples scripts
	12. Uso de FastAPI para servir datos manipulados por aplicaciones
Streamlit	13. Configuración de
la página	14. A dónde ir desde aquí
	14.1 Documentación, blog y foro
de	Streamlit
de Streamlit	14.2 Componentes
	15. Sobre el autor

1. Cómo ver este tutorial mejor en su dispositivo

En algunos navegadores de escritorio, el texto y las imágenes de esta página pueden parecer inicialmente pequeños. Presione 'CTRL' y '+' en su teclado para ampliar el documento para que llene su pantalla. Si está leyendo

en un dispositivo móvil, pellizque y haga zoom para expandir cualquier imagen / código. Si tiene una pantalla muy pequeña, gire su teléfono hacia un lado a lo largo para rotar la vista al modo horizontal. Eso proporcionará la vista más amplia posible del contenido del documento. Es posible que desee actualizar su navegador después de rotarlo en modo horizontal, de modo que cada fragmento de texto llene la pantalla de la manera más ordenada posible. También puede optar por ver en modo 'simplificado', si su navegador móvil sugiere esa opción.

2. Requisitos previos

Debes tener una comprensión fundamental de Python para comenzar este tutorial. Si no ha escrito código Python (o cualquier otro) antes, es posible que desee comenzar con pythonanvil.com. Ese mini curso gratuito de 200+ páginas enseña todo lo que necesita saber para crear varias docenas de aplicaciones web completas, utilizando otro marco de Python puro ([Anvil](#)), sin necesidad de conocimientos previos de programación. Muchas de las ideas de aplicaciones y ejemplos de código genérico de Python explicados en ese minicurso de Anvil se comparten con este tutorial, por lo que es un requisito previo perfecto para principiantes para este texto.

3. ¿Qué es Streamlit?

Streamlit es una biblioteca de Python que permite a los desarrolladores crear rápidamente interfaces de usuario interactivas (diseños de páginas web) con *muy poco código*. Puede usar Streamlit junto con cualquier sistema de base de datos disponible en Python, para formar un kit de herramientas de desarrollo web *de pila completa* fácil de usar, especialmente adecuado para proyectos internos donde la productividad y las iteraciones de desarrollo rápidas son una prioridad. Una parte importante de este tutorial se centrará en la creación de aplicaciones de gestión de datos CRUD mediante la integración de Streamlit con comandos SQL y el sistema de base de datos SQLite. Otra gran sección del tutorial muestra patrones de código Streamlit en la práctica utilizando pequeñas aplicaciones de ejemplo.

El siguiente ejemplo es una aplicación completa que *permite a los usuarios* codificar, *rear*, *enviar* *eliminar* entradas de base de datos con formularios Streamlit. Estas 22 líneas son la lista *de código completa* necesaria para compilar y configurar cada parte de la aplicación, incluida la interfaz de usuario y todas las interacciones de la base de datos:

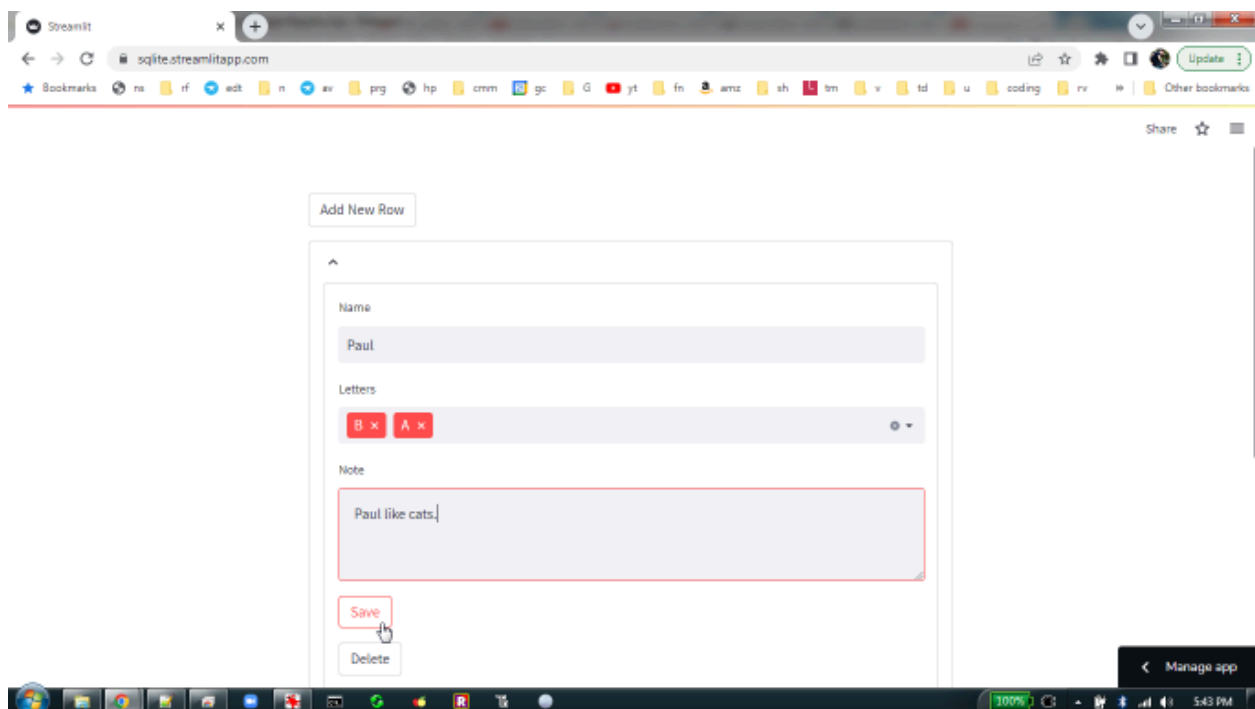
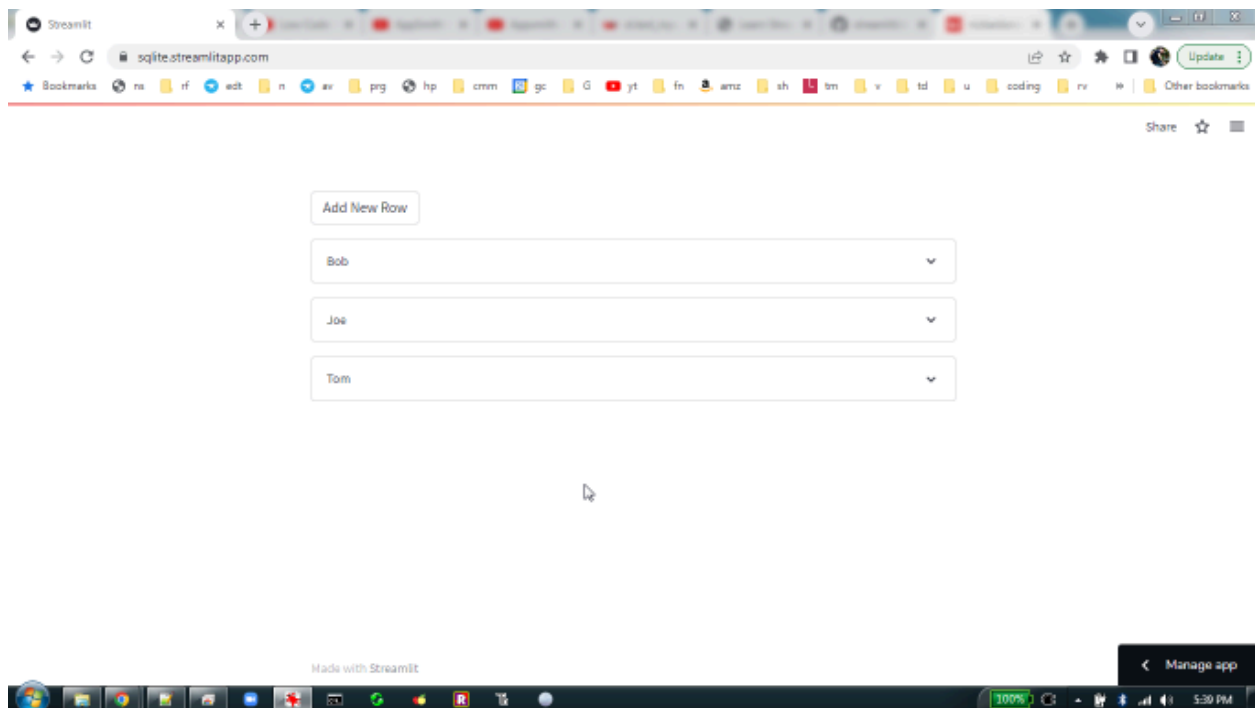
```
import streamlit as st, ast, sqlite3
con=sqlite3.connect('db.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')

if st.button('Add New Row'):
    cur.execute('INSERT INTO db(name, letters, note) VALUES(?,?,?)', ('','[]',''))
    con.commit()

for row in cur.execute('SELECT rowid, name, letters, note FROM db ORDER BY name'):
    with st.expander(row[1]):
        with st.form(f'ID-{row[0]}'):
            name=st.text_input('Name', row[1])
            letters=st.multiselect('Letters', ['A', 'B', 'C'], ast.literal_eval(row[2]))
            note=st.text_area('Note', row[3])
            if st.form_submit_button('Save'):
                cur.execute(
                    'UPDATE db SET name=?, letters=?, note=? WHERE name=?;',
                    (name, str(letters), note, str(row[1]))
                )
                con.commit() ; st.experimental_rerun()
            if st.form_submit_button("Delete"):
                cur.execute(f'DELETE FROM db WHERE rowid="{row[0]}"')
                con.commit() ; st.experimental_rerun()
```

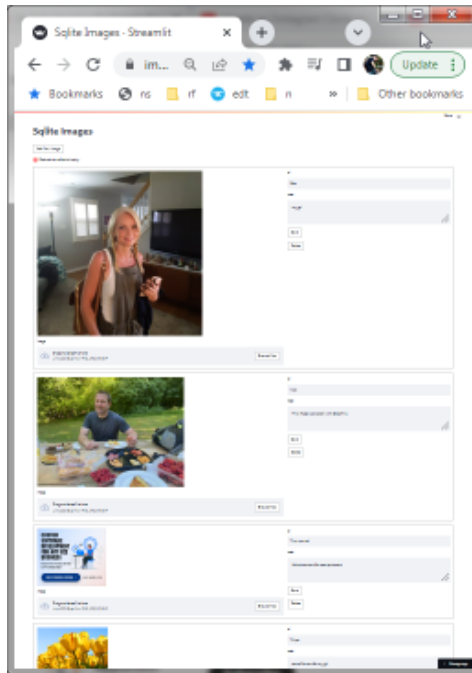
Aquí hay una versión en vivo de la aplicación, que está alojada de forma gratuita con Streamlit Cloud:

[Ejemplo de CRUD mínimo de SQLite](#)



Aquí hay una versión ligeramente modificada de la aplicación anterior, que almacena y muestra imágenes e información relacionada:

[Ejemplo de imagen Sqlite mínima](#)



4. ¿Por qué usar Streamlit?

Streamlit está diseñado para ser utilizado como una herramienta de presentación para científicos de datos, por lo que su mayor fortaleza es la integración de todas las mejores bibliotecas de visualización, análisis, aprendizaje automático e inteligencia artificial de Python: Pandas, Numpy, Matplotlib, Tensorflow, OpenCV, PyTorch, Theano, Plotly, Altair, Bokeh, GraphViz, PyDeck, Vegalite, así como toda la biblioteca estándar y cualquier otra biblioteca/herramienta en el mundo de Python.

Streamlit *solo requiere Python* para crear interfaces web enriquecidas (no se necesita absolutamente HTML/CCS/JS) y permite a los desarrolladores usar cualquier código Python de forma nativa en el back-end para realizar tareas complejas de gestión de datos, IA, aprendizaje automático, consumo de API y conectividad en minutos.

Puede ejecutar aplicaciones web Streamlit/Python inmediatamente en iPhone, iPad, Android, Windows, Mac, Linux, Chromebook, etc., *sin ninguna instalación* en esos dispositivos. Los únicos requisitos para ejecutar Streamlit son una instalación básica de Python en una máquina Windows, Mac o Linux local (es decir, conectada a WIFI), o en una plataforma de servidor en la nube, una instalación rápida de pip de la biblioteca Streamlit y cualquier editor de texto.

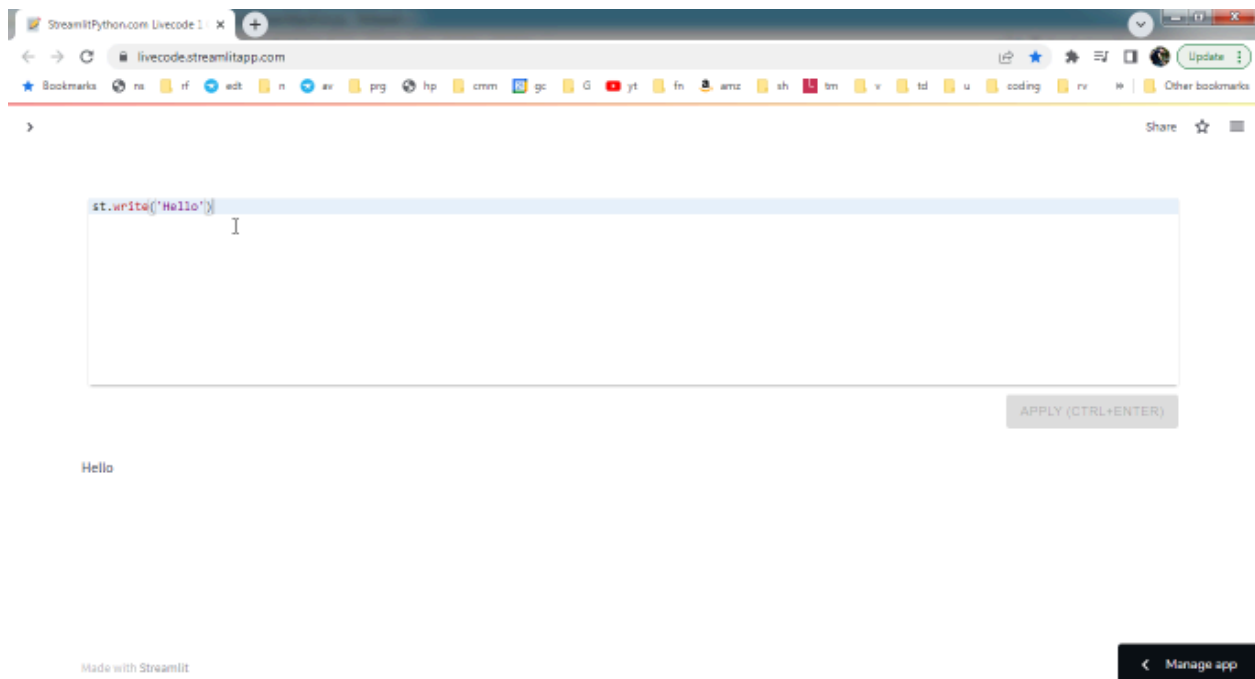
Streamlit es gratuito y de código abierto, y está respaldado por una empresa bien financiada. [Streamlit Cloud](#) proporciona una solución alojada gratuita y conveniente que permite una implementación instantánea, sin tareas que consumen mucho tiempo ni costos típicamente asociados con la publicación de aplicaciones web. La integración de GIT está integrada directamente en la plataforma de alojamiento gratuito de Streamlit (las aplicaciones alojadas se actualizan automáticamente cada vez que se confirman los archivos del repositorio de Github de un proyecto). No hay bloqueo de proveedor de ningún tipo: puede alojar aplicaciones en el sitio localmente o utilizando AWS, Google Cloud, Azure, Digital Ocean, Heroku, etc.

5. Primeros pasos

Para comenzar con este tutorial, haga clic en el siguiente enlace:

livecode.streamlitapp.com

El editor de código en vivo anterior se ejecuta en su navegador e inmediatamente muestra los resultados del código Streamlit editado. Incluso puede ejecutar este editor en el navegador web de cualquier dispositivo móvil moderno:



(El editor de código en vivo utilizado aquí fue creado en Streamlit por Sam Dobson, basado en el trabajo de Okld, utilizando el editor JS Ace).

5.1 Hola Mundo

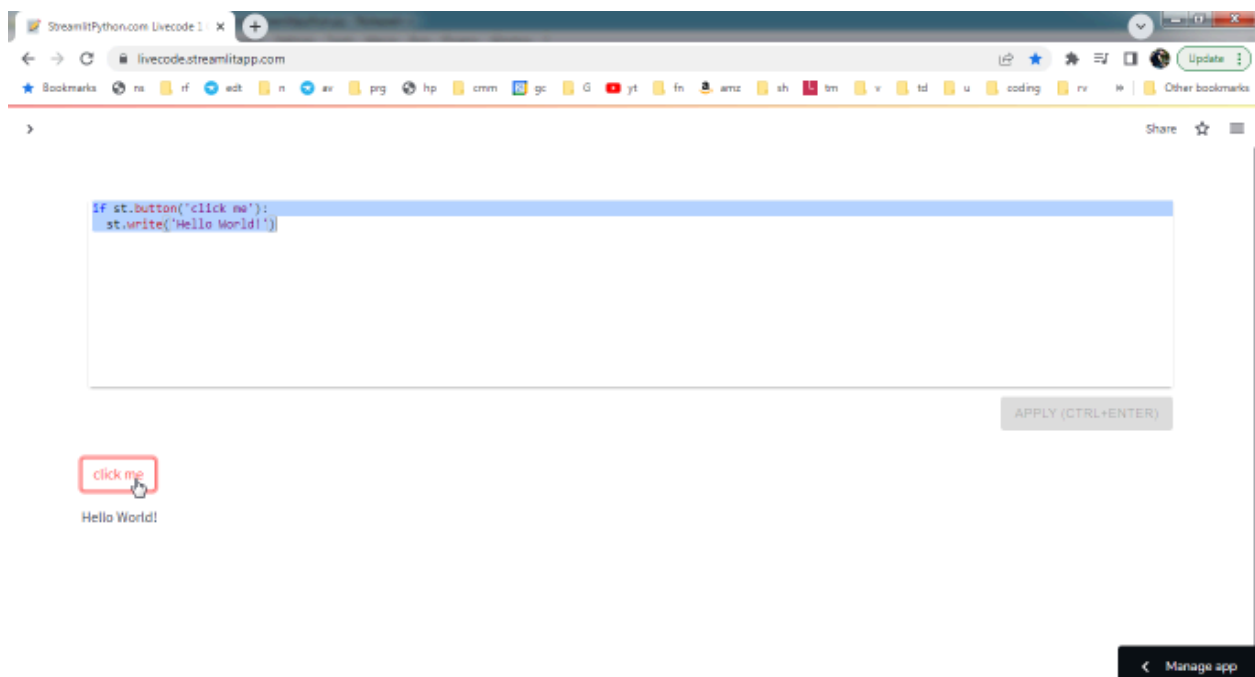
En el [editor de código en vivo](#), reemplace el código predeterminado:

```
st.write('Hello')
```

Con el siguiente código:

```
if st.button('click me'):  
    st.write('Hello World!')
```

Haga clic en el botón 'Aplicar' en la esquina inferior derecha del editor (o presione las teclas CTRL + Enter en su teclado) y verá una nueva actualización de la aplicación debajo del editor. Haz clic en el botón "Haz clic en mí" que acaba de aparecer y verás aparecer el mensaje "¡Hola mundo!":



Eso es todo: solo 2 líneas de código y tenemos una aplicación web interactiva.

Observe que Streamlit utiliza el método 'st.write()' para mostrar valores. El método 'st.button()' se utiliza para presentar un widget de botón en el que se puede hacer clic. Se utiliza una evaluación 'if' para ejecutar código Python cada vez que se hace clic en el botón.

IMPORTANTE: de forma predeterminada en las aplicaciones Streamlit, *todo el script se vuelve a ejecutar cada vez que un usuario interactúa con cualquier tipo de widget de entrada*. Cubriremos más sobre cómo funciona esto, más adelante en el tutorial.

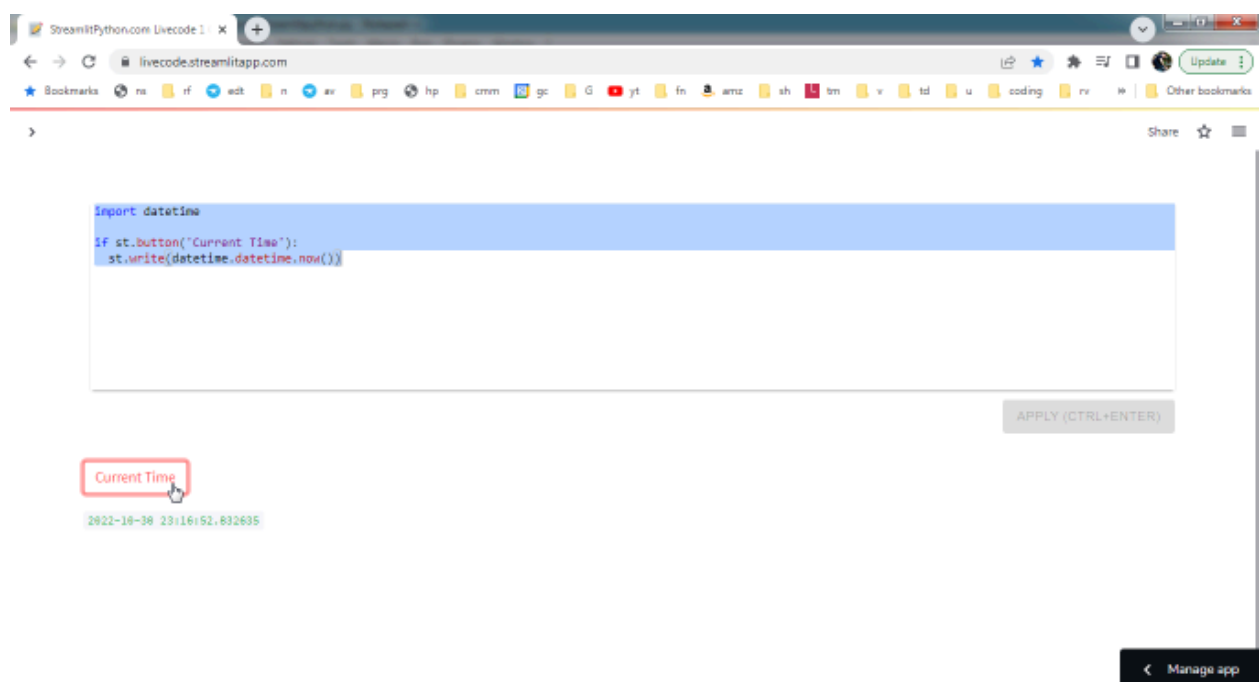
Si recibe algún error al ejecutar un script en el editor de livecode, haga clic en el botón "Administrar aplicación" en la esquina inferior derecha de la pantalla para ver información detallada de depuración.

5.2 Uso de la biblioteca estándar de Python con Streamlit

Ahora reemplace el código en el editor con lo siguiente (luego haga clic en 'Aplicar' o presione CTRL+Enter):

```
import datetime

if st.button('Current Time'):
    st.write(datetime.datetime.now())
```



Con ese fragmento de código, ahora estamos usando una función de la biblioteca estándar de Python, *directamente dentro del mismo archivo de script único que produce la interfaz del navegador web*. El código front-end y back-end está todo en ese archivo. En lugar de estar atascados en una consola de escritorio imprimiendo texto, los usuarios pueden interactuar con este código Python en un teléfono móvil / tableta o cualquier otro dispositivo que tenga un navegador.

Agárrense a sus asientos, apenas estamos comenzando...

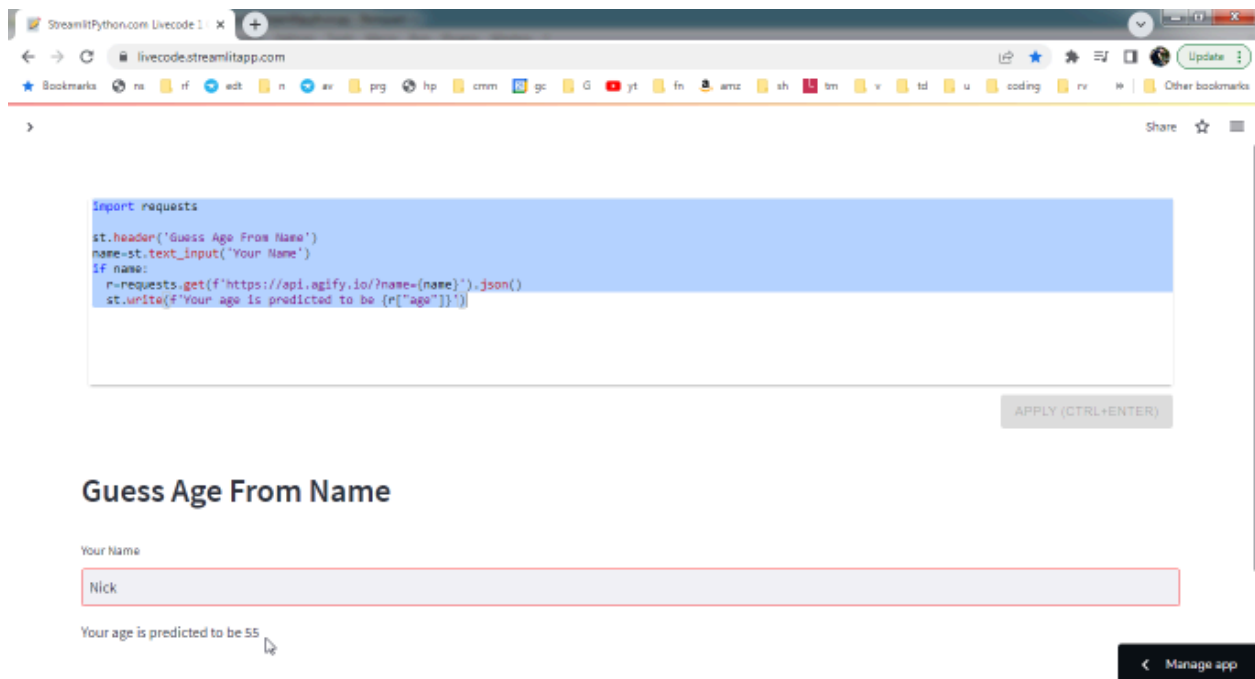
5.3 Interacción con las API web

A continuación, ejecute este código:

```
import requests

st.header('Guess Age From Name')
name=st.text_input('Your Name')
if name:
    r=requests.get(f'https://api.agify.io/?name={name}') . json()
    st.write(f'Your age is predicted to be {r["age"]}')

```



El ejemplo anterior utiliza la biblioteca de 'solicitudes' de Python para leer datos de la API web en www.agify.io (la biblioteca de solicitudes se ha preinstalado en el servidor en la nube que ejecuta nuestro editor de código en vivo, que solo toma un momento para `pip install`, cada vez que desee ejecutar Streamlit en su propio entorno de producción de Python).

Tenga en cuenta que el método '`st.text_input()`' en el código anterior se usa para mostrar un widget de entrada de campo de texto. *El valor escrito por el usuario en el widget `text_input` se almacena en la variable 'name'*. Al igual que con el botón en el ejemplo anterior, usamos una evaluación 'if' aquí para ejecutar código Python cada vez que el usuario cambia esa variable (ingresada en el widget `text_input`). Tenga en cuenta que cada vez que el usuario interactúa con el widget `st.text_input()`, *se vuelve a ejecutar todo el script, utilizando ese nuevo valor de variable asignado al contenido de texto del widget*.

Observe también el widget '`st.header()`', que agrega un título de texto grande al diseño de la aplicación.

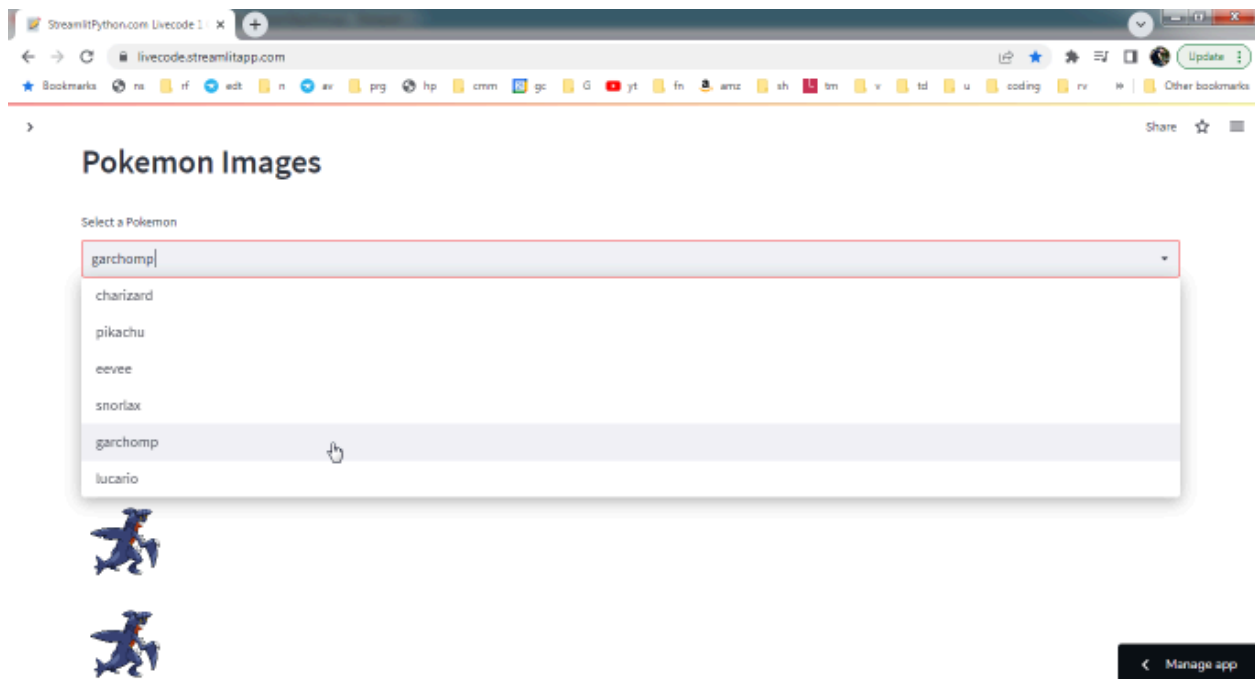
5.4 Un ejemplo de la API de Pokémon

Ahora veamos otro ejemplo de API web para familiarizarnos más con los métodos Streamlit. Tenga en cuenta que en el [editor de livecode](#) que estamos utilizando, la biblioteca Streamlit se importa de forma predeterminada. En cualquier otro entorno, deberá importar Streamlit, por lo que todos los ejemplos de este tutorial en el futuro incluirán Streamlit en las importaciones de bibliotecas. Streamlit se importa convencionalmente como 'st':

```
import streamlit as st, requests

st.header('Pokemon Images')
mypokemon=['charizard','pikachu','eevee','snorlax','garchomp','lucario']
pokemon=st.selectbox('Select a Pokemon', mypokemon)
if pokemon:
    r=requests.get(f'https://pokeapi.co/api/v2/pokemon/{pokemon}') .json()
    for img in r['sprites'].values():
        if img is not None:
            if str(img)[-4:]==' .png':
                st.image(img)
```

El ejemplo anterior utiliza un widget '`st.selectbox()`' para permitir a los usuarios elegir personajes de Pokémon de una lista desplegable. Se emplea un bucle 'for' y algunas evaluaciones 'if' para analizar el json devuelto por la API de pokeapi.co, y se usa un widget '`st.image()`' para mostrar cualquier imagen vinculada que se encuentre en los resultados:



Es importante darse cuenta de que la mayor parte del ejemplo anterior es solo *código básico de Python*, exactamente como lo que usaría en un REPL de Python de línea de comandos o en scripts simples de consola de escritorio. En estos ejemplos, solo estamos agregando algunos métodos 'st.()' para obtener información del usuario y mostrar los resultados en la página web. El código típico de Python se ejecuta *en el servidor* donde se ejecuta Streamlit y los widgets de la interfaz de usuario de Streamlit *se ejecutan en el navegador*. Streamlit se encarga de transportar todos los datos entre la máquina del servidor y los navegadores web del cliente, de forma transparente. Es extraordinariamente simple de aprender.

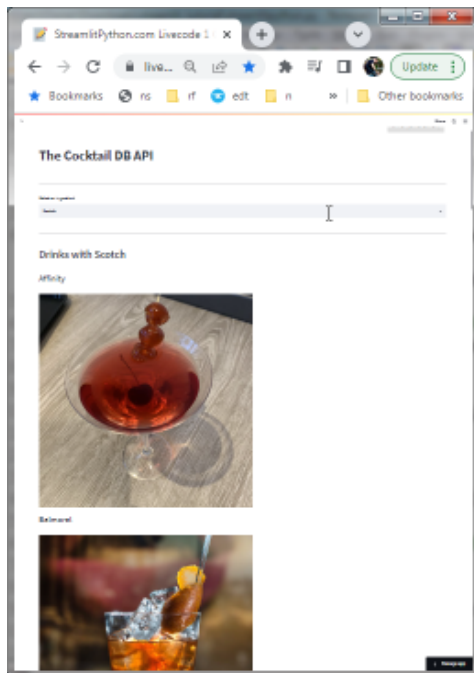
5.5 Ejemplo de API de base de datos de cócteles

Aquí hay otro ejemplo de API similar para aclarar cómo incorporar Streamlit con el típico código Python de 'estilo consola':

```
import streamlit as st, requests

drinks=requests.get(
    f'https://thecocktaildb.com/api/json/v1/1/list.php?i=list'
).json()
ingredients=[]
for drink in drinks['drinks']:
    ingredients.append(drink['strIngredient1'])

st.title('The Cocktail DB API')
st.markdown('---')
selected=st.selectbox('Select an Ingredient', ingredients)
st.markdown('---')
if selected:
    st.header(f'Drinks with {selected}')
    r=requests.get(
        f'https://thecocktaildb.com/api/json/v1/1/filter.php?i={selected}'
    ).json()
    for drink in r['drinks']:
        st.subheader(drink['strDrink'])
        st.image(drink['strDrinkThumb'])
```

En el ejemplo anterior, Streamlit se encarga de ejecutar todo el código del widget en el navegador del usuario y el código del servidor en la computadora donde se ejecuta Streamlit, sin que el desarrollador necesite diferenciar nada sobre dónde ocurren esos procesos en la pila web:

```
##### THIS CODE RUNS ON THE SERVER #####
import streamlit as st, requests

drinks=requests.get(
    f'https://thecocktaildb.com/api/json/v1/1/list.php?i=list'
).json()
ingredients=[]
for drink in drinks['drinks']:
    ingredients.append(drink['strIngredient1'])

##### THIS CODE RUNS IN THE BROWSER #####
st.title('The Cocktail DB API')
st.markdown('---')
selected=st.selectbox('Select an Ingredient', ingredients)
st.markdown('---')
if selected:
    st.header(f'Drinks with {selected}')

##### THIS CODE RUNS ON THE SERVER #####
r=requests.get(
    f'https://thecocktaildb.com/api/json/v1/1/filter.php?i={selected}'
).json()
for drink in r['drinks']:

##### THIS CODE RUNS IN THE BROWSER #####
    st.subheader(drink['strDrink'])
    st.image(drink['strDrinkThumb'])
```

Esta es la magia de Streamlit: la maquinaria de pila completa necesaria para lo que tradicionalmente son interacciones cliente-servidor complicadas, se crea con un archivo de código, utilizando un proceso de pensamiento lineal sencillo y familiar. Puede pasar variables en un *solo script*, casi como si todo el código se estuviera ejecutando en una sola máquina, y Streamlit obtiene los datos de ida y vuelta entre la computadora del servidor y los navegadores del cliente, sin ningún esfuerzo por parte del desarrollador. Esto es bastante diferente de la rutina tradicional de desarrollo full stack.

5.6 Más widgets de entrada y salida de Streamlit

Aquí hay una demostración de muchos elementos de entrada y visualización de la interfaz de usuario de Streamlit. Puede encontrar la documentación de la API para todos estos widgets en <https://docs.streamlit.io/library/api-reference/widgets> (marque esa página como favorita):

```

import streamlit as st

st.title('Title')
st.header('Header')
st.subheader('Subheader')
st.caption('Caption')
st.code('print("this is some code")')
st.text('Text')
st.markdown('- *Markdown*')
st.latex('\sum_{k=0}^{n-1} ar^k')
st.dataframe(pd.DataFrame({'a':[1, 2, 3], 'b':['A', 'B', 'C']}))
st.table({'a':[1, 2, 3], 'b':['A', 'B', 'C']})
st.metric('Temp', '75', '5')
st.metric('Wind', '9', '-4')
st.json({'a':[1, 2, 3], 'b':['A', 'B', 'C']})
st.button('Button')
st.download_button('Download Button', b'asdf')
st.checkbox('Checkbox')
st.radio('Radio', [1, 2, 3])
st.selectbox('Selectbox', ['a', 'b', 'c'])
st.multiselect('Multiselect', ['a', 'b', 'c'])
st.slider('Slider')
st.select_slider('Select Slider', ['a', 'b', 'c'])
st.text_input('Text Input')
st.number_input('Number Input')
st.text_area('Text Area')
st.date_input('Date Input')
st.time_input('Time Input')
st.file_uploader('File Uploader')
st.camera_input('Camera')
st.color_picker('Color Picker')
st.image('tulips.jpg')
st.audio('audio.mp3')
st.video('video.mp4')
st.sidebar.selectbox('Menu', ['a', 'b', 'c'])
col1, col2 = st.columns([1, 2])
col1.text_input('Thinner Column')
col2.text_input('Thicker Column')
tab1, tab2 = st.tabs(['TAB 1', 'TAB 2'])
tab1.text_area('text in tab 1')
tab2.date_input('date in tab 2')
st.expander('Expander')
st.container()
placeholder = st.empty()
placeholder.text('Hide this placeholder container')
if st.button('Hide'): placeholder.empty()
st.progress(35)
st.spinner('Spinner')
if st.checkbox('Balloons', False):
    st.balloons()
if st.checkbox('Snow', False):
    st.snow()
st.error('Error')
st.warning('Warning')
st.info('Info')
st.success('Success')
st.exception(RuntimeError('This is a fake error.'))
# st.form('')
# st.form_submit_button('')
import numpy as np
df=pd.DataFrame(np.random.randn(20, 3), columns=['a', 'b', 'c'])
st.line_chart(df)
st.area_chart(df)
st.bar_chart(df)
treemap=pd.read_csv('trees2.csv')
st.write(treemap)
st.map(treemap)

```

StreamlitPython.com Livecode

livecode.streamlitapp.com

na rf edit n av prg hp cmm gc G yt fn amz sh tm v td u coding rv Other bookmarks

Share ☆

Title

Header

Subheader

Caption

```
print("this is some code")
```

Text

- Markdown

$$\sum_{k=0}^{n-1} ar^k$$

	a	b

Manage app

StreamlitPython.com Livecode

livecode.streamlitapp.com

na rf edit n av prg hp cmm gc G yt fn amz sh tm v td u coding rv Other bookmarks

Share ☆

Temp

75

↑ 5

Wind

9

↓ -4

	a	b
0	1	A
1	2	B
2	3	C

		a	b
0		1	A
1		2	B
2		3	C

Manage app

StreamlitPython.com Livecode x

livecode.streamlitapp.com

Bookmarks na rf edit n av prg hp cmm gc G yt fn amz sh tm v td u coding rv Other bookmarks

Share ☆

```
{
  "a": [
    0: 1
    1: 2
    2: 3
  ]
  "b": [
    0: "A"
    1: "B"
    2: "C"
  ]
}
```

Button

Download Button

Checkbox

Radio

☒ 1

☐ 2

Manage app

StreamlitPython.com Livecode x

livecode.streamlitapp.com

Bookmarks na rf edit n av prg hp cmm gc G yt fn amz sh tm v td u coding rv Other bookmarks

Share ☆

Selectbox

a

Multiselect

Choose an option

Slider

0 100

Select Slider

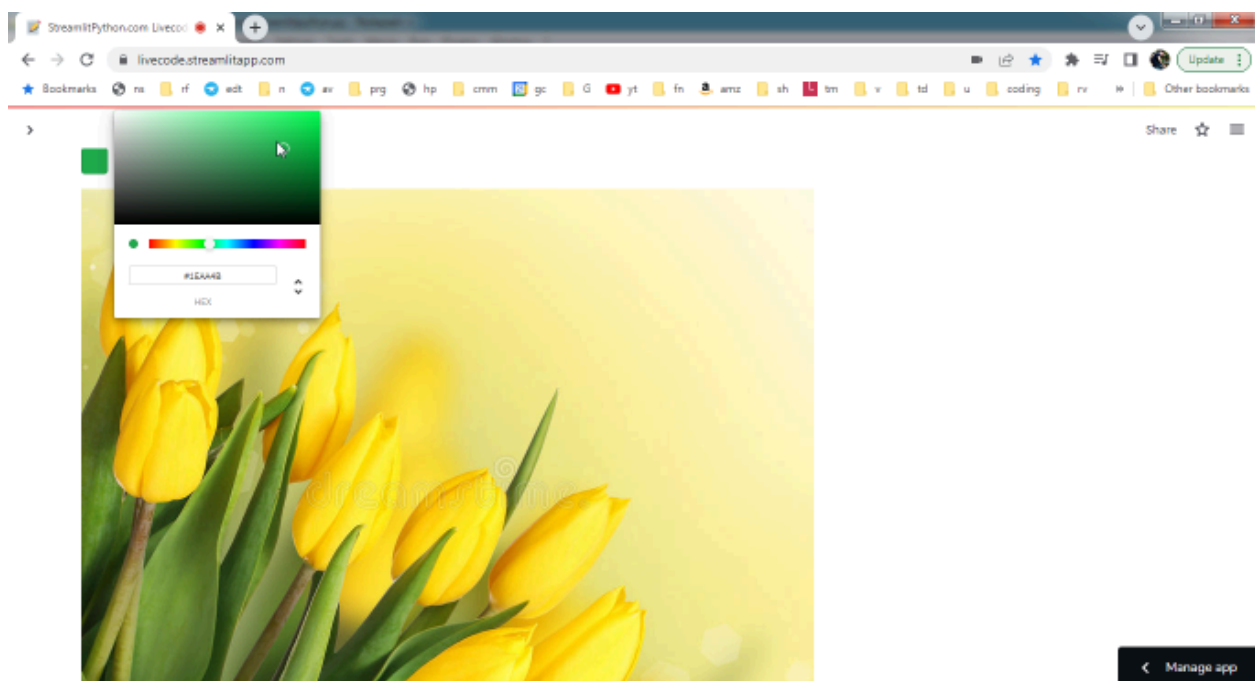
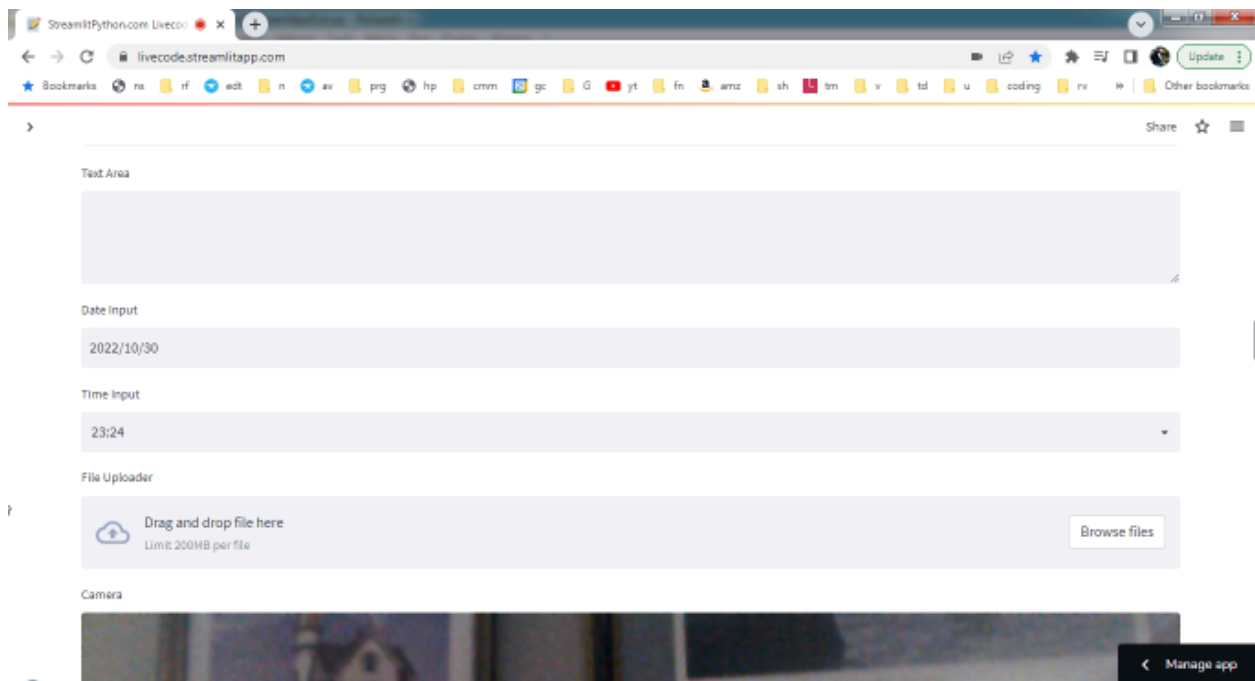
a c

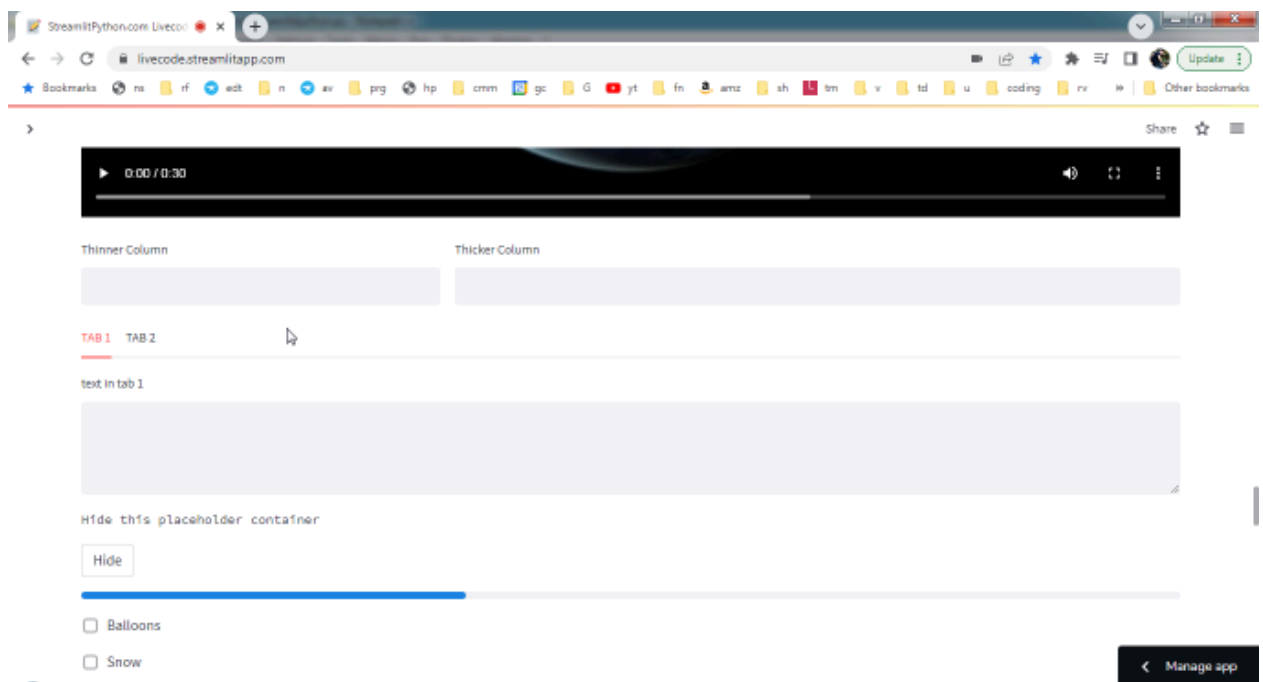
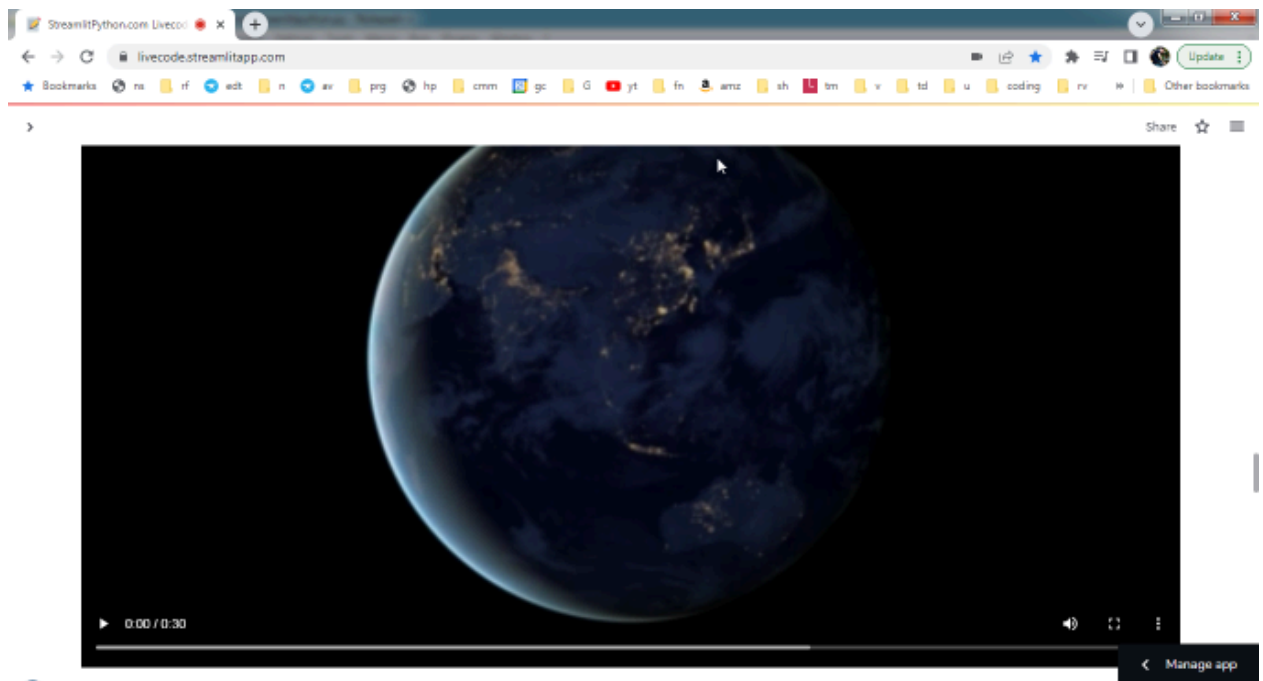
Test Input

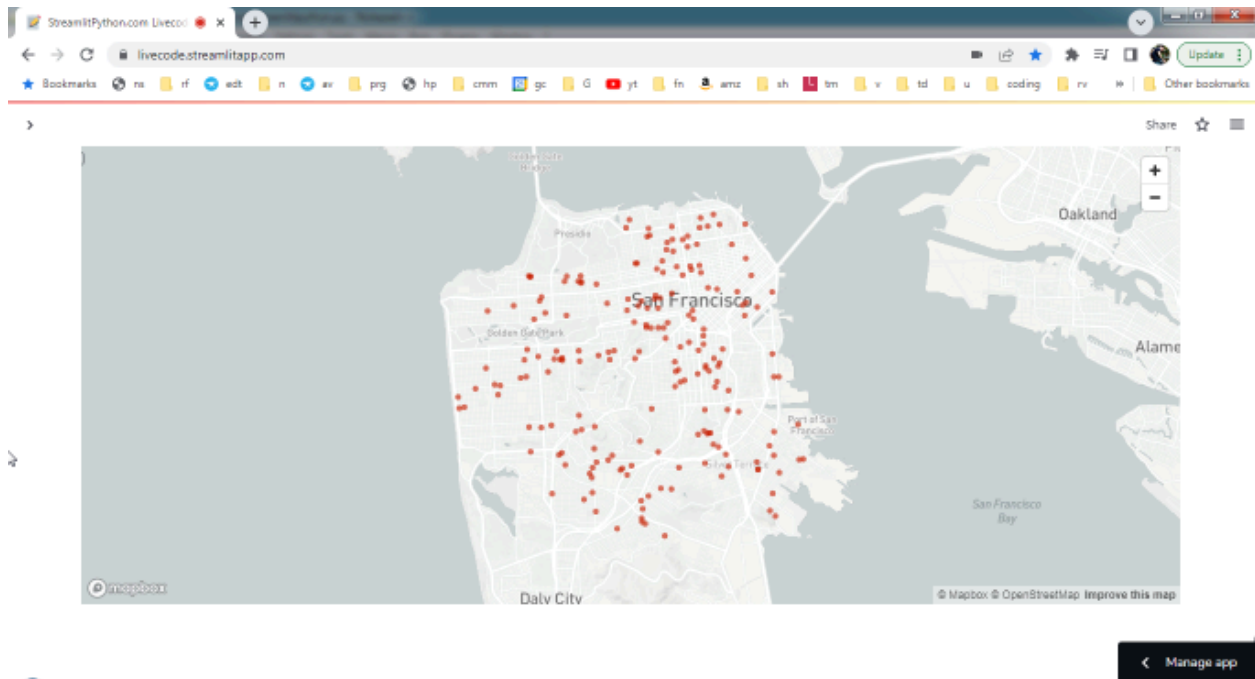
Number Input

0.00

Manage app





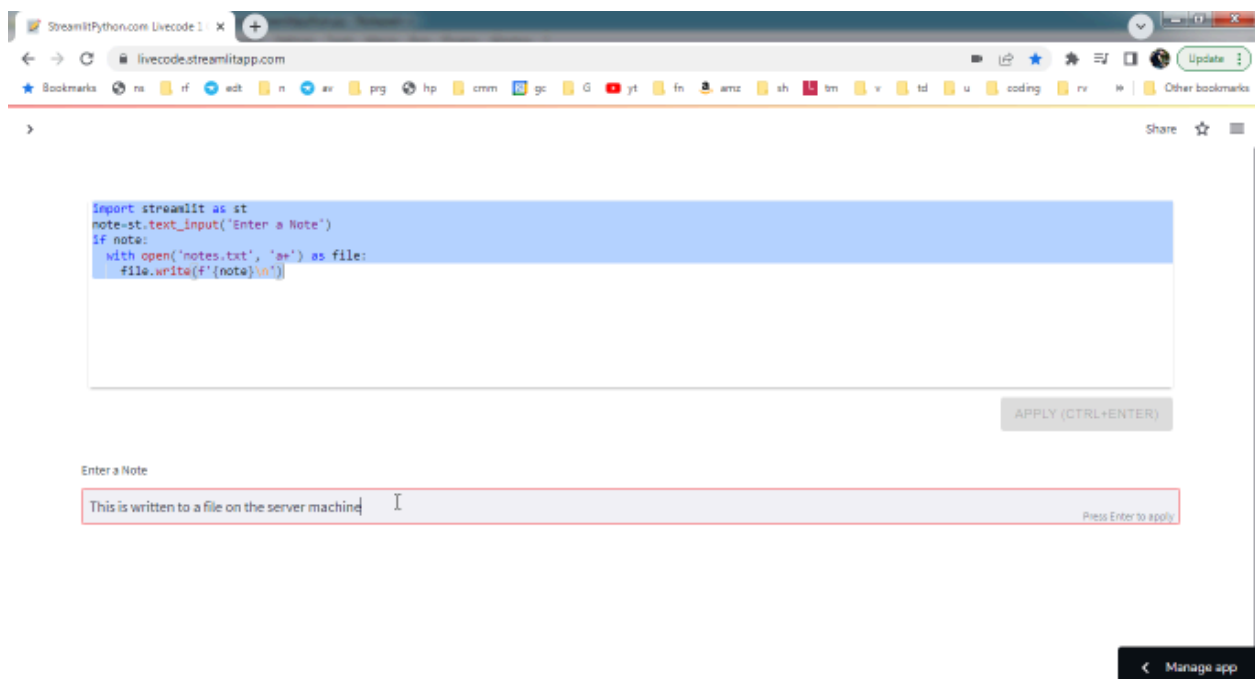


5.7 Simplemente asigne y pase variables

La mayoría de los métodos anteriores se ejecutan en el navegador para mostrar elementos de la interfaz de usuario, pero algunos, como `file_uploader`, `download_button`, `imagen`, `audio` y `video`, transfieren datos entre el servidor y el cliente de forma inherente.

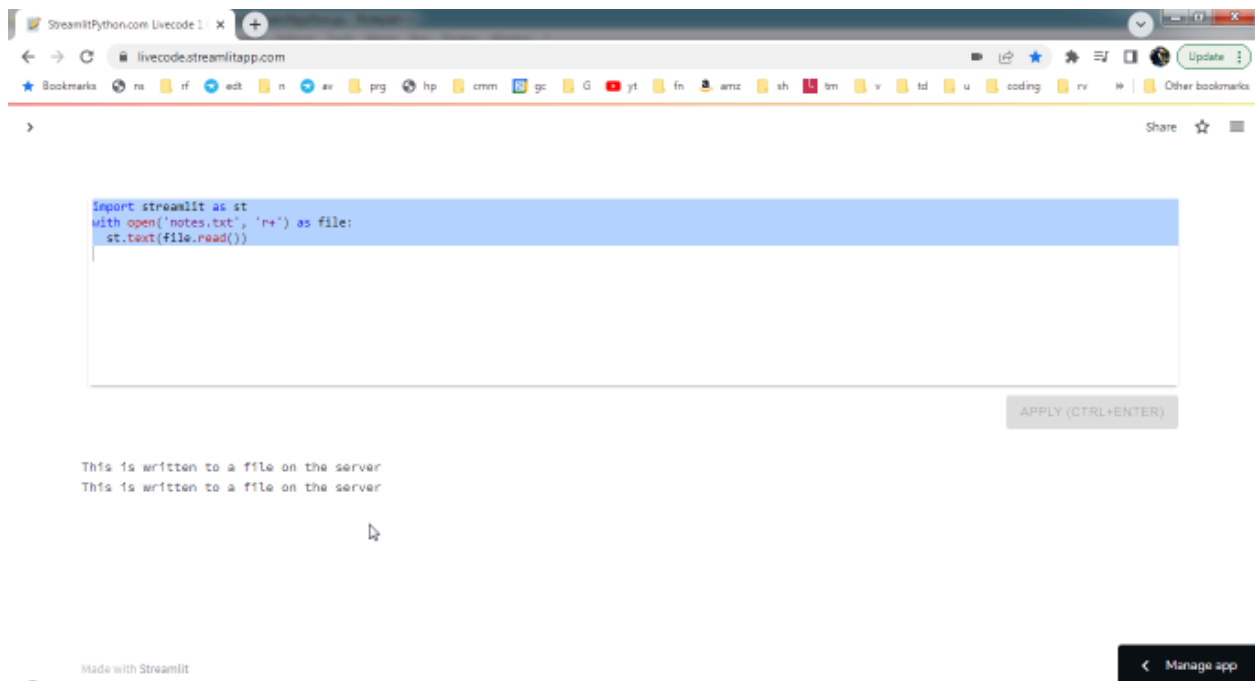
A todos los widgets de entrada se les pueden asignar variables, y esas variables se pueden usar en el código Python normal que se ejecuta en el servidor. Por ejemplo, el código siguiente obtiene un valor del usuario, en el navegador, y luego escribe esos datos en el disco duro del servidor, utilizando algunas operaciones de archivo típicas en código Python de 'estilo consola':

```
import streamlit as st
note=st.text_input('Enter a Note')
if note:
    with open('notes.txt', 'a+') as file:
        file.write(f'{note}\n')
```



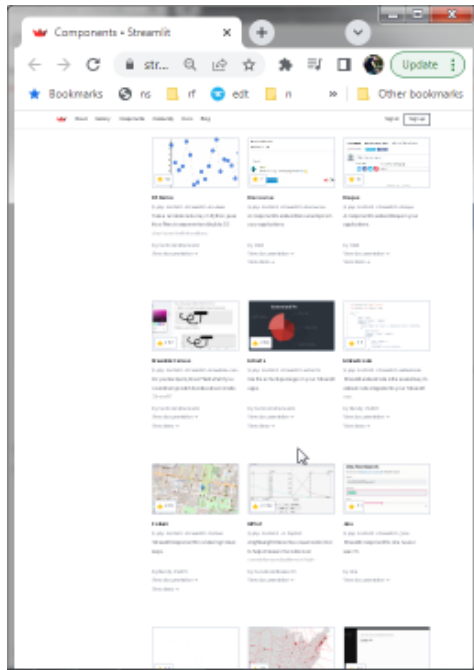
Viceversa también es cierto: puede asignar una variable al código que se ejecuta en el servidor y mostrar esos datos en el navegador, simplemente usando la variable en un método de widget Streamlit:


```
import streamlit as st
with open('notes.txt', 'r+') as file:
    st.text(file.read())
```



Eso es solo la punta del iceberg. Streamlit tiene soporte incorporado para muchas de las bibliotecas de visualización y gráficos de Python más populares, así como un mundo de potentes componentes gratuitos creados por la comunidad, que permiten interesantes funciones de interfaz de usuario / diseño / diseño (incluidas cuadrículas de datos complejas), procesamiento avanzado de gráficos / audio / video, mapeo geográfico, videoconferencia en tiempo real, integraciones con aplicaciones de terceros como Discourse y Disqus, y mucho más, todo con muy poco código y todo usando *Python puro*.

<https://streamlit.io/components>



6. Flujo de programa optimizado

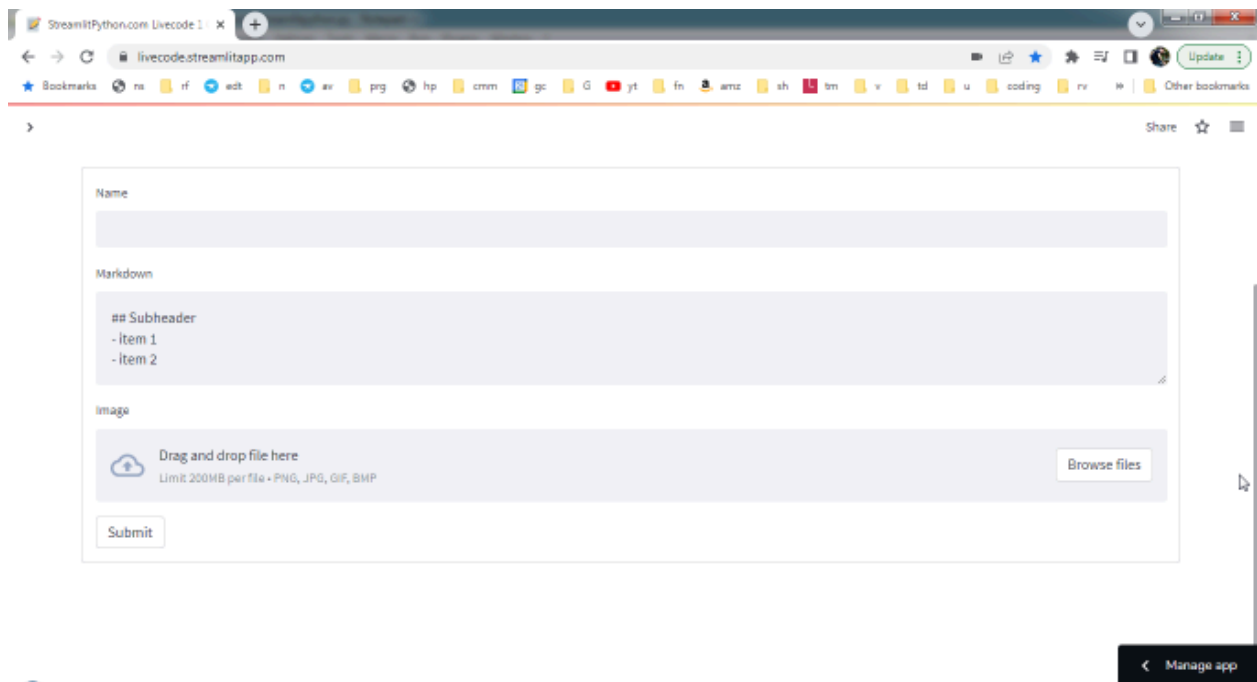
Como ha visto, cada vez que un usuario interactúa con un widget de entrada de Streamlit, la respuesta predeterminada es que todo el *script* se vuelve a evaluar con el nuevo valor ingresado en ese widget. Por lo general, esto ahorra mucho trabajo a los desarrolladores de Streamlit en la asignación de funciones de devolución de llamada y en la administración del estado, pero puede generar algunos desafíos.

6.1 Distribución

En la mayoría de las aplicaciones, habrá situaciones en las que no desee que Streamlit vuelva a evaluar todo el script cada vez que se introduzca un valor en un widget, o situaciones en las que necesite guardar los valores introducidos anteriormente, por ejemplo. La más común de esas situaciones se produce cuando se utilizan formularios, donde solo se desea procesar un grupo de valores introducidos una vez que el usuario los ha introducido:

```
import streamlit as st

with st.form('My Form', clear_on_submit=True):
    name=st.text_input('Name')
    mrkdwn=st.text_area('Markdown', '## Subheader\n- item 1\n- item 2')
    file=st.file_uploader('Image', ['png', 'jpg', 'gif', 'bmp'])
    if st.form_submit_button('Submit'):
        st.markdown(f'## {name}\n{mrkdwn}')
        st.image(file)
```

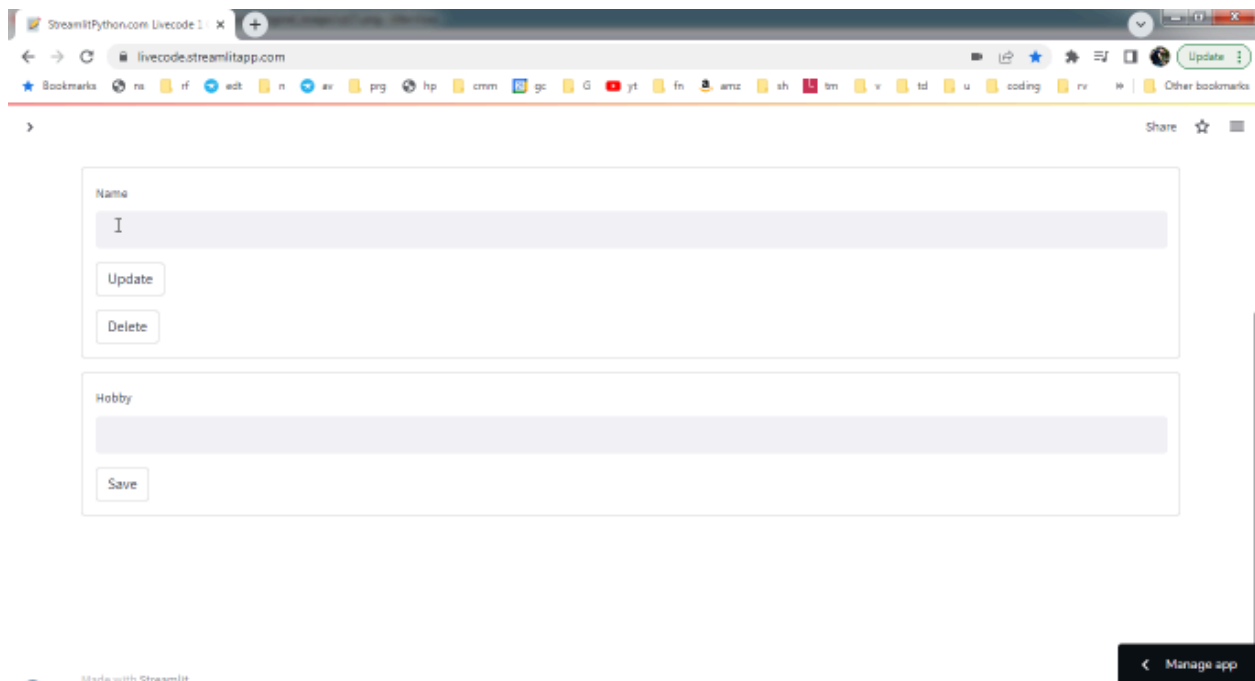


Los valores de las variables de un formulario no se evalúan hasta que se hace clic en el widget `st.form_submit_button()`. En ese momento, todo el script se vuelve a ejecutar con esos valores de entrada.

Puede incluir tantos formularios como necesite en una página e incluir tantos widgets `st.form_submit_button()` como sea necesario en cada formulario, para procesar datos de diferentes maneras. Cada formulario debe tener un nombre único:

```
import streamlit as st

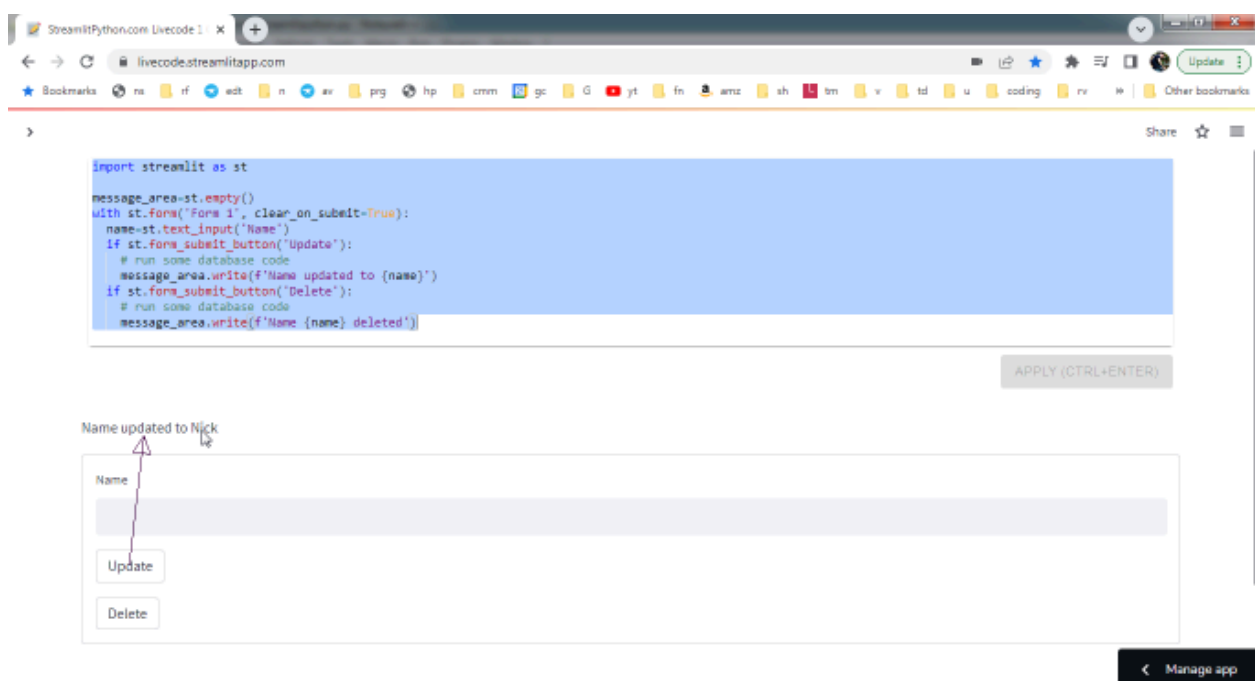
with st.form('Form 1', clear_on_submit=True):
    name=st.text_input('Name')
    if st.form_submit_button('Update'):
        # run some database code
        st.write(f'Name updated to {name}')
    if st.form_submit_button('Delete'):
        # run some database code
        st.write(f'Name {name} deleted')
with st.form('Form 2', clear_on_submit=True):
    hobby=st.text_input('Hobby')
    if st.form_submit_button('Save'):
        # run some database code
        st.write(f'{hobby} saved')
```



Puede notar en el ejemplo anterior que la notificación 'Nombre actualizado a {name}' mostrada por `st.write()` aparece en una ubicación extraña en la pantalla, directamente debajo del botón de envío del formulario, porque ahí es donde aparece en el código. *Los scripts de Streamlit siempre se ejecutan de arriba a abajo y se vuelven a evaluar cada vez que se ingresan nuevos valores de entrada en la aplicación en ejecución.* Para colocar esa notificación de actualización en otro lugar del diseño, podemos usar `st.empty()` de la siguiente manera:

```
import streamlit as st

message_area=st.empty()
with st.form('Form 1', clear_on_submit=True):
    name=st.text_input('Name')
    if st.form_submit_button('Update'):
        # run some database code
        message_area.write(f'Name updated to {name}')
    if st.form_submit_button('Delete'):
        # run some database code
        message_area.write(f'Name {name} deleted')
```



Hay una serie de [otros contenedores](#) que se pueden usar para colocar widgets en la pantalla en los diseños deseados. Aquí hay dos formas sintácticas diferentes de colocar widgets en contenedores `st.columns()`:

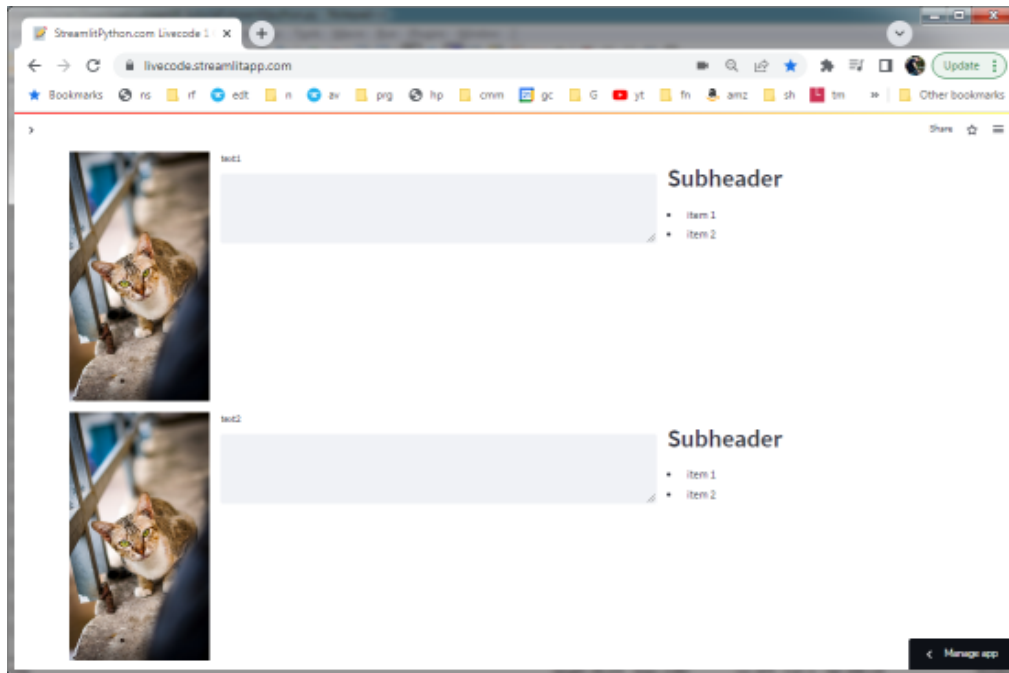
```

import streamlit as st

col1, col2, col3=st.columns([1,3,2])
col1.image("https://static.streamlit.io/examples/cat.jpg")
col2.text_area('text1')
col3.markdown('## Subheader\n- item 1\n- item 2')

col4, col5, c6=st.columns([1,3,2])
with col4:
    st.image("https://static.streamlit.io/examples/cat.jpg")
with col5:
    st.text_area('text2')
with col6:
    st.markdown('## Subheader\n- item 1\n- item 2')

```



El primer método anterior es útil cuando desea especificar el contenedor de columnas en el que se debe colocar un solo widget, especialmente cuando se crean instancias de varios widgets a partir de secciones separadas de un script.

El segundo método es útil cuando se desea colocar un mayor número de widgets en una columna determinada, todos en un solo bloque de código contiguo, utilizando un administrador de contexto 'with', con sangría debajo de la etiqueta de columna prevista.

6.2 Valores persistentes y almacenamiento en caché

El siguiente script *no* muestra un incremento, como cabría esperar, porque Streamlit vuelve a ejecutar el script cada vez que se hace clic en el botón. La variable 'total' se restablece a cero cada vez que se vuelve a ejecutar el script:

```

total=0
if st.button('+'):
    total+=1
    st.write(total)

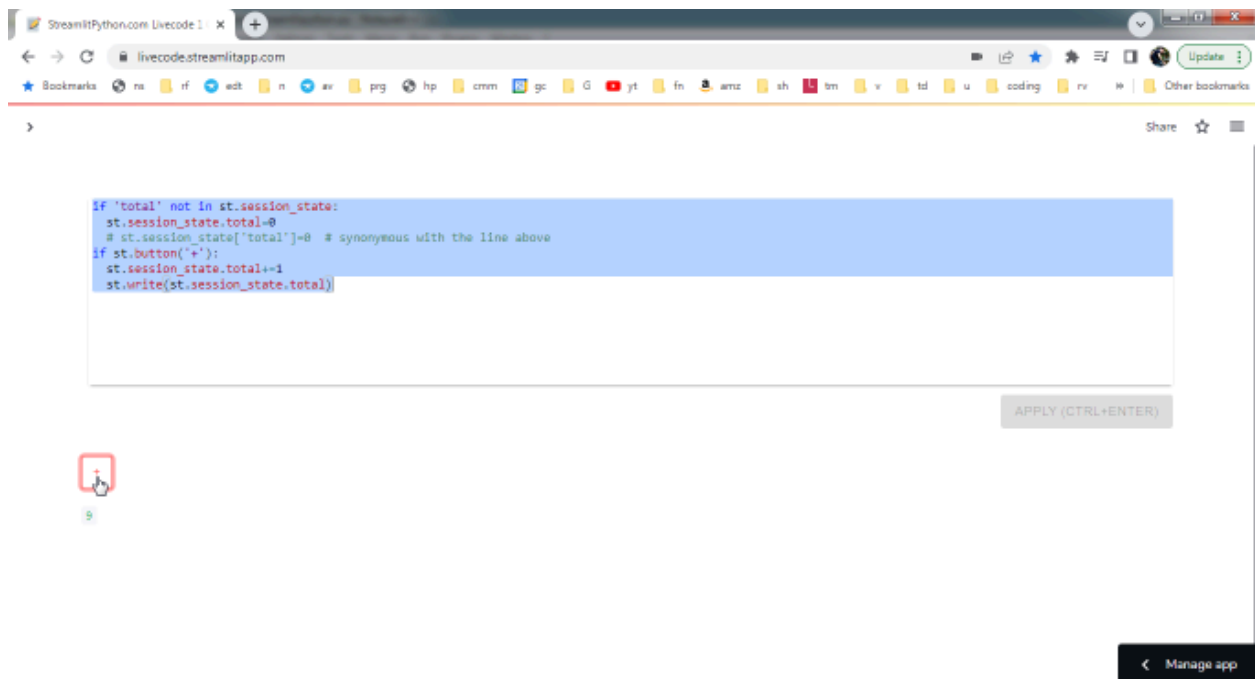
```

Para almacenar valores persistentes durante cada repetición de un script de Streamlit, use `st.session_state()`:

```

if 'total' not in st.session_state:
    st.session_state.total=0
    # st.session_state['total']=0 # synonymous with the line above
if st.button('+'):
    st.session_state.total+=1
    st.write(st.session_state.total)

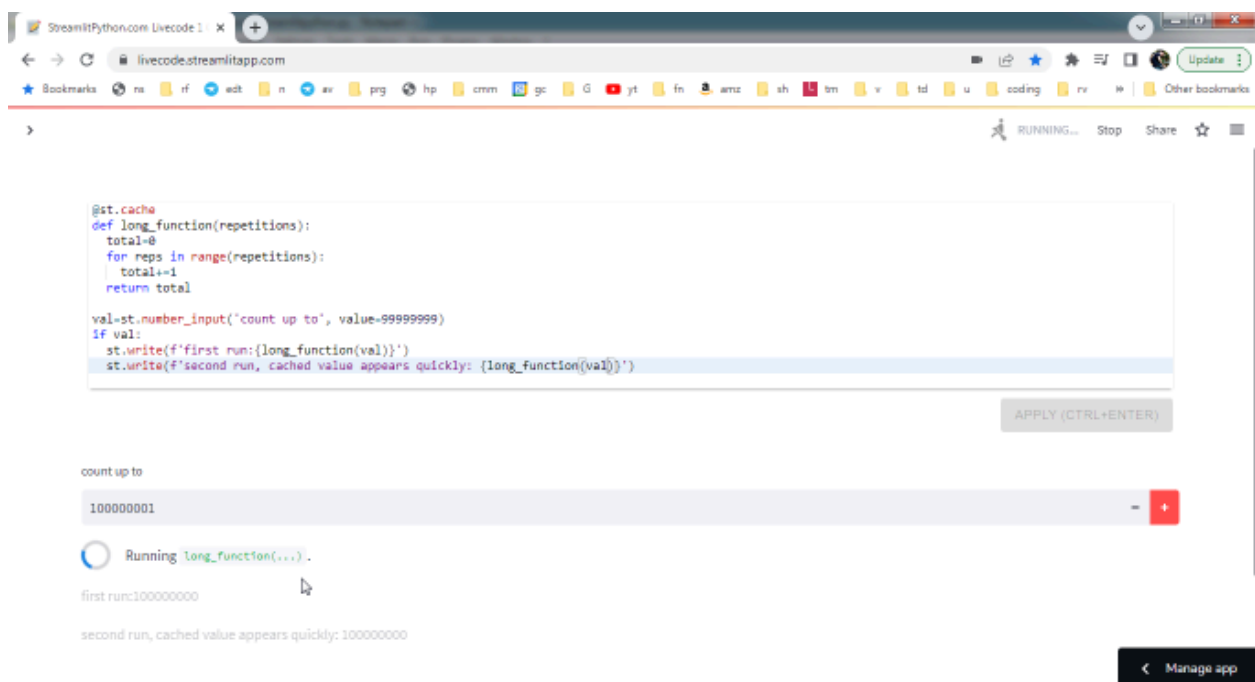
```



Para guardar los resultados de los cálculos de larga duración, de modo que esas rutinas costosas no se vuelvan a evaluar con cada interacción de entrada del usuario, use `st.cache()`:

```
@st.cache
def long_function(repetitions):
    total=0
    for reps in range(repetitions):
        total+=1
    return total

val=st.number_input('count up to', value=99999999)
if val:
    st.write(f'first run:{long_function(val)}')
    st.write(f'second run, cached value appears quickly: {long_function(val)}')
```



Tenga en cuenta que la primera ejecución de cualquier número grande seleccionado requiere un cálculo largo. Cada vez que esa función se vuelve a ejecutar con el mismo valor seleccionado, el valor devuelto aparece inmediatamente, ya que la función de cálculo utiliza el valor de caché guardado para cuando el argumento de entrada es idéntico.

Para borrar los valores de caché guardados, haga clic en el menú de hamburguesas en la esquina superior derecha de la interfaz de usuario de Streamlit y seleccione 'Borrar caché'.

Puede obtener más información sobre el almacenamiento en caché de valores en <https://docs.streamlit.io/library/api-reference/performance>.

7. Creación de aplicaciones básicas de base de datos CRUD con Streamlit

(NOTA: Si tiene experiencia trabajando con SQL en Python, puede hojear las siguientes 2 secciones de este tutorial. Reutilizaremos algunas líneas de estas secciones repetidamente).

Las operaciones de 'Crear/Leer/Actualizar/Eliminar' (CRUD) son un componente clave de la mayoría de las aplicaciones útiles, especialmente las que se utilizan para la gestión de datos empresariales (contabilidad, inventario, facturación, nómina, punto de venta, programación, comunicaciones, gestión de proyectos, gestión de propiedades, etc.). Streamlit y Python son una gran combinación para crear la funcionalidad de la aplicación web CRUD de forma rápida y sencilla.

Para este tutorial, mantendremos las cosas simples y usaremos algunos comandos SQL básicos para almacenar y recuperar datos utilizando el sistema de base de datos SQLite. SQLite está integrado en Python, por lo que no hay nada adicional que instalar (viene gratis como parte integrada de todas las distribuciones modernas de Python). Hay muchas opciones de Python que se pueden usar para migrar de SQLite a otros RDBMS, siempre que necesite capacidad de nivel empresarial. Los ORM (mapeadores relacionales de objetos) como SQLAlchemy eliminan la necesidad de código SQL y ayudan a unificar la conectividad entre sistemas de bases de datos populares como PostgreSQL, MySQL, MSSQL, Oracle y otros.

7.1 Un manual rápido de Python SQL

Hay muchos tutoriales en línea que explican cómo usar SQL en Python, pero para nuestros propósitos, cubriremos solo algunos fragmentos fundamentales que le permiten crear una tabla de base de datos, luego insertar, eliminar, actualizar y recuperar filas de datos.

Primero, para usar SQLite, debe importar 'sqlite3' desde la biblioteca estándar de Python:

```
import sqlite3
```

Para conectarse con una base de datos SQLite, usará el método `.connect` de la biblioteca `sqlite3`, especificando el nombre de un archivo de base de datos SQLite. Si este archivo no existe, se creará. Asignaremos la variable 'con' para referirnos a esta conexión:

```
con=sqlite3.connect('db.db')
```

A continuación, usará el método `.cursor` de la base de datos conectada para enviar comandos SQL a la base de datos conectada. Asignaremos la variable 'cur' para referirnos a este cursor:

```
cur=con.cursor()
```

A continuación, el método `.execute` del cursor se usa para enviar una cadena que contiene comandos SQL. El primer comando SQL que necesitaremos, para crear una tabla en la base de datos, es 'CREATE TABLE'. Incluiremos la cláusula 'IF NOT EXISTS', de modo que esta operación solo se produzca si la tabla aún no se ha creado. Las tablas se pueden considerar como filas y columnas de información, similares a una cuadrícula de hoja de cálculo. La definición de los nombres de las columnas y el tipo de datos que puede contener cada columna se denomina "esquema" de la tabla. Llamaremos a esta tabla 'db' y crearemos un esquema con 3 columnas llamadas 'name', 'letters' y 'note', todas las cuales almacenarán valores de texto:

```
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')
```

Para insertar valores de datos en la tabla creada anteriormente, utilice el comando 'INSERT INTO'. El código siguiente agrega una nueva fila de datos a las columnas 'name', 'letters' y 'note' de la tabla 'db', utilizando 3 marcadores de posición '?', rellenos con los valores ('', '[]', ''):

```
cur.execute('INSERT INTO db(name, letters, note) VALUES(?,?,?)', ('', '[]', ''))
```

Siempre que se complete una operación, use el método `.commit` del objeto de conexión de base de datos para guardar los datos en la base de datos (antes de hacer esto, los comandos SQL simplemente se almacenan en la memoria RAM y los resultados aún no se guardan en el archivo de base de datos):

```
con.commit()
```

Para actualizar los valores de una fila de la tabla de la base de datos (también llamada 'registro'), utilice el comando SQL 'UPDATE'. El código siguiente actualiza el registro vacío que creamos anteriormente, con algunos valores significativos:

```
cur.execute(
    'UPDATE db SET name=?, letters=?, note=? WHERE name=?;',
    ('John Smith', '["A", "B"]', 'Has a popular name', ''))
con.commit()
```

Para ver los registros de la base de datos, utilice el comando SQL 'SELECT'. El siguiente código devuelve las columnas 'rowid', 'name', 'letters' y 'note' de las tablas 'db', y las ordena en orden por valores en la columna 'name':

```
cur.execute('SELECT rowid, name, letters, note FROM db ORDER BY name')
```

Tenga en cuenta que la columna 'rowid' es una 'clave', generada automáticamente e incrementada automáticamente por SQLite, de modo que cada fila tiene un identificador único. Esto se hace para que las filas se puedan diferenciar y hacer referencia a ellas específicamente, incluso si hay valores duplicados en otros campos. El rowid se usa para asegurarse de que realiza operaciones en una fila deseada, por ejemplo, para eliminar la entrada correcta para una persona determinada llamada John Smith, cuando potencialmente hay varias filas con el nombre 'John Smith' en el campo de nombre.

Para eliminar registros, utilice el comando 'ELIMINAR'. El siguiente código elimina el registro con rowid 1 de la tabla 'db':

```
cur.execute('DELETE FROM db WHERE rowid="1";')
con.commit()
```

Con esos pocos ejemplos de código SQL, tenemos suficientes herramientas para realizar un trabajo realmente útil.

7.2 (Des)Serialización de listas de Python y otras estructuras de datos en SQLite

Dependiendo del sistema de base de datos, encontrará soporte para una amplia variedad de tipos de datos (fechas y horas, imágenes y datos binarios, valores json, etc.). Algunos sistemas de bases de datos, como PostgreSQL, tienen soporte integrado para valores nativos de Python, como listas y diccionarios, en forma de tipos de datos json. SQLite es tan simple como las bases de datos, por lo que para guardar estructuras de datos como listas, diccionarios, conjuntos y tuplas, simplemente podemos serializar los datos en cadenas, usando la función `str()` de Python. Al leer cadenas de una base de datos, podemos usar el método `ast.literal_eval()` para convertir las estructuras de datos guardadas en tipos de datos de lista, diccionario, conjunto y tupla funcionales.

`ast.literal_eval()` es mucho más seguro que usar la función `eval()` de Python, ya que solo funciona para evaluar estructuras de datos (a diferencia del código ejecutable). Esto ayuda a desinfectar el código malicioso almacenado por los usuarios en la base de datos. No desea que el código ejecutable que ha sido almacenado por un usuario, por ejemplo, pueda borrar el contenido de su disco duro o proporcionar involuntariamente derechos de administrador a un usuario. ¡Eso es potencialmente bastante fácil de hacer si alguna vez usa `eval()` sobre los valores almacenados en una base de datos!

Un patrón típico que podemos usar para almacenar listas de Python en una base de datos se ve así (observe la función `str()` aplicada a la lista de 'letras', en la declaración de actualización de SQL):

```
name='John Doe'
letters=['A', 'B', 'C']
note='John is from Ohio'
cur.execute(
    'UPDATE db SET name=?, letters=?, note=? WHERE name=?;',
    (name, str(letters), note, name))
con.commit()
```

Podemos recuperar los datos guardados de la base de datos utilizando un patrón como este (observe el método `ast.literal_eval()` aplicado al campo 'letras', recuperado por la declaración SQL select):

```
for row in cur.execute('SELECT rowid, name, letters, note FROM db ORDER BY name'):
    name=row[1]
    letters=ast.literal_eval(row[2])
    note=row[3]
```

También usaremos métodos de fecha y hora para formatear las fechas correctamente en los próximos ejemplos

Cuando se trabaja con otros sistemas de bases de datos, es preferible usar los tipos de datos nativos empleados por cada RDBMS en particular, en lugar de serializar los tipos de datos en cadenas, como se requiere para SQLite.

7.3 Cableado de sentencias SQL con widgets de interfaz de usuario Streamlit

Para comenzar a crear aplicaciones de base de datos con interfaces de usuario de Streamlit que aparecen en un navegador, solo necesitamos mezclar algunos métodos de widgets de Streamlit en el código SQL. Aquí están todas las importaciones, así como todo el código de configuración de la base de datos que necesitamos, tomado literalmente de las secciones anteriores de este tutorial:

```
import streamlit as st, sqlite3, ast
con=sqlite3.connect('db.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')
```

Para agregar una nueva fila a la base de datos, podemos usar un widget `st.button()`. Este código SQL también se toma literalmente de los ejemplos del tutorial anterior:

```
if st.button('Add New Row'):
    cur.execute('INSERT INTO db(name, letters, note) VALUES(?,?,?)', (',' '[]', ''))
    con.commit()
```

Una de las formas más sencillas de mostrar las filas devueltas por una consulta a la base de datos es utilizar un widget `st.expander()`. El `st.expander()` de Streamlit es un poco como un widget de acordeón: pliega una sección de la página web para ocultar diseños de cualquier complejidad, y se expande para mostrar todo el diseño seleccionado cuando un usuario hace clic en él.

En este primer ejemplo, usaremos un bucle `for` para generar widgets `st.expander()` para cada fila devuelta por una consulta de base de datos (un comando SQL `select`). Dentro de cada `st.expander()`, colocaremos un widget `st.form()`, que se utiliza para mostrar un grupo de widgets `st.text_input()`, `st.multiselect()` y `st.text_area()`, cada uno de los cuales presenta un valor seleccionado de la base de datos (de nuevo, el código SQL `SELECT` aquí se toma literalmente de la sección anterior de este tutorial):

```
for row in cur.execute('SELECT rowid, name, letters, note FROM db ORDER BY name'):
    with st.expander(row[1]):
        with st.form(f'ID-{row[0]}'):
            name=st.text_input('Name', row[1])
            letters=st.multiselect('Letters', ['A', 'B', 'C'], ast.literal_eval(row[2]))
            note=st.text_area('Note', row[3])
```

Tenga en cuenta que los widgets `st.form()` anteriores deben tener claves únicas (el primer argumento del método `st.form()`), por lo que se usa una cadena formateada para concatenar 'ID-' con el valor 'rowid' de la fila de la base de datos que se muestra:

```
st.form(f'ID-{row[0]}')
```

7.4 Terminar una aplicación CRUD básica: agregar botones para actualizar y eliminar

Podemos agregar widgets `st.form_submit_button()` al código de formulario anterior para ejecutar declaraciones de actualización y eliminación de SQL sobre cualquier valor de fila mostrado. Tenga en cuenta que los botones del código siguiente están contenidos en formularios separados creados por el bucle `for`, por lo que cada valor de fila al que se hace referencia en las instrucciones `update` y `delete` (ejecutadas cuando el usuario hace clic en cualquier widget de botón) hace referencia al único registro particular que se muestra en cada formulario. Eso significa que al hacer clic en cualquier botón 'Actualizar' o 'Eliminar' se realizan esas operaciones en la fila de valores de la base de datos que se muestra en ese formulario específico:


```

for row in cur.execute('SELECT rowid, name, letters, note FROM db ORDER BY name'):
    with st.expander(row[1]):
        with st.form(f'ID-{row[0]}'):
            name=st.text_input('Name', row[1])
            letters=st.multiselect('Letters', ['A', 'B', 'C'], ast.literal_eval(row[2]))
            note=st.text_area('Note', row[3])
            if st.form_submit_button('Save'):
                cur.execute(
                    'UPDATE db SET name=?, letters=?, note=? WHERE name=?;',
                    (name, str(letters), note, str(row[1]))
                )
                con.commit()
                st.experimental_rerun()
            if st.form_submit_button("Delete"):
                cur.execute(f'DELETE FROM db WHERE rowid="{row[0]}"')
                con.commit()
                st.experimental_rerun()

```

Tenga en cuenta que el método `st.experimental_rerun()` se ejecuta después de las operaciones de actualización y eliminación. Esto solo actualiza toda la visualización de la página con valores de consulta recién actualizados leídos de la base de datos.

Lo creas o no, eso es todo lo que necesitamos para crear una aplicación web completa de base de datos CRUD con Streamlit. Esta es la lista de código completa (el mismo ejemplo presentado en la introducción de este tutorial):

```

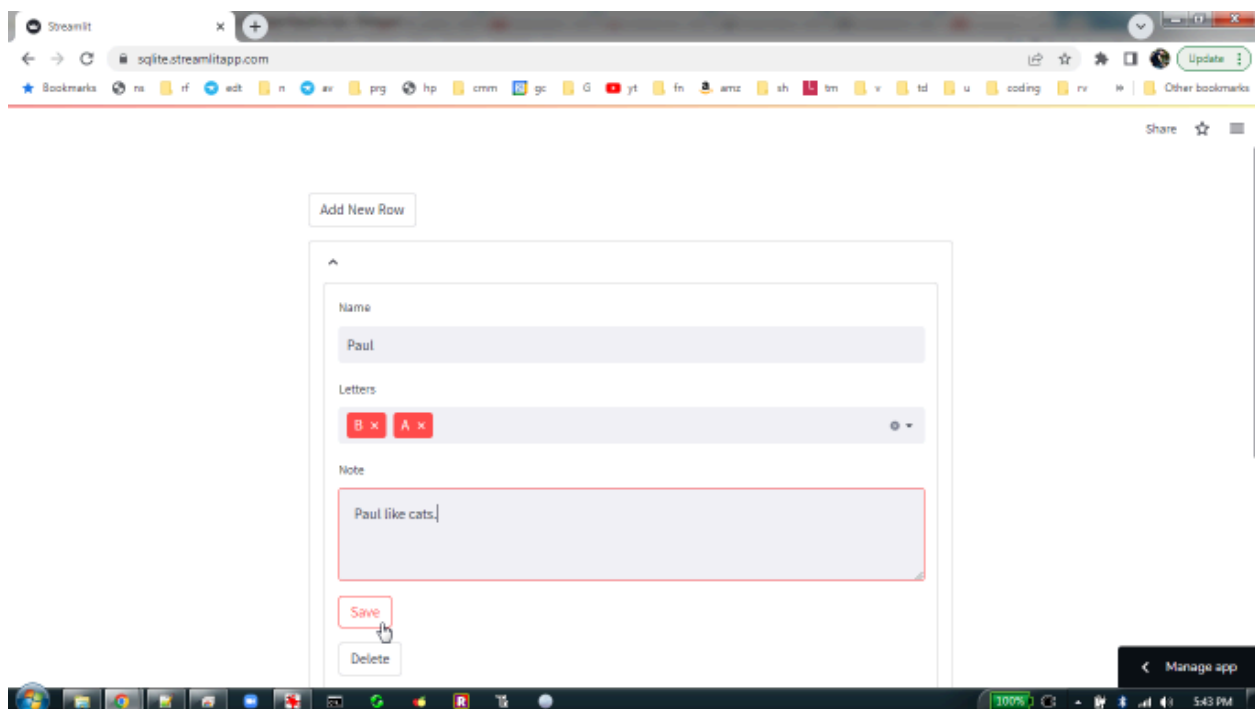
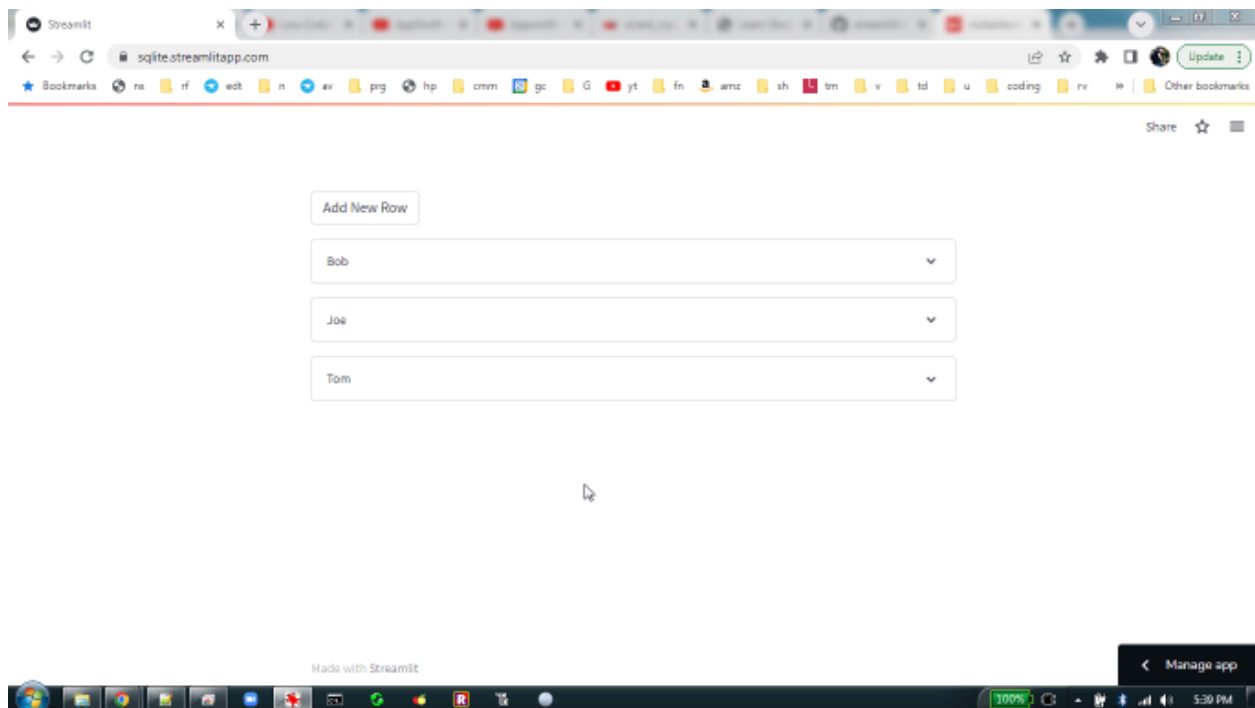
import streamlit as st, ast, sqlite3
con=sqlite3.connect('db.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')

if st.button('Add New Row'):
    cur.execute('INSERT INTO db(name, letters, note) VALUES(?,?,?)', (',', '[]', ''))
    con.commit()

for row in cur.execute('SELECT rowid, name, letters, note FROM db ORDER BY name'):
    with st.expander(row[1]):
        with st.form(f'ID-{row[0]}'):
            name=st.text_input('Name', row[1])
            letters=st.multiselect('Letters', ['A', 'B', 'C'], ast.literal_eval(row[2]))
            note=st.text_area('Note', row[3])
            if st.form_submit_button('Save'):
                cur.execute(
                    'UPDATE db SET name=?, letters=?, note=? WHERE name=?;',
                    (name, str(letters), note, str(row[1]))
                )
                con.commit() ; st.experimental_rerun()
            if st.form_submit_button("Delete"):
                cur.execute(f'DELETE FROM db WHERE rowid="{row[0]}"')
                con.commit() ; st.experimental_rerun()

```

¡Pega ese ejemplo en el editor de código en vivo en livecode.streamlitapp.com, para verlo ejecutarse!



7.5 Trabajar con imágenes, sonidos y videos - Aplicación de base de datos de imágenes

Podemos tomar el ejemplo anterior como modelo, cambiar un poco el esquema de la base de datos, usar algunos widgets para cargar y mostrar imágenes, y tenemos un pequeño sistema de base de datos perfectamente utilizable para manejar fotos:

```
import streamlit as st, sqlite3

con=sqlite3.connect('pics.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS pics(id TEXT, img BLOB, note TEXT)')

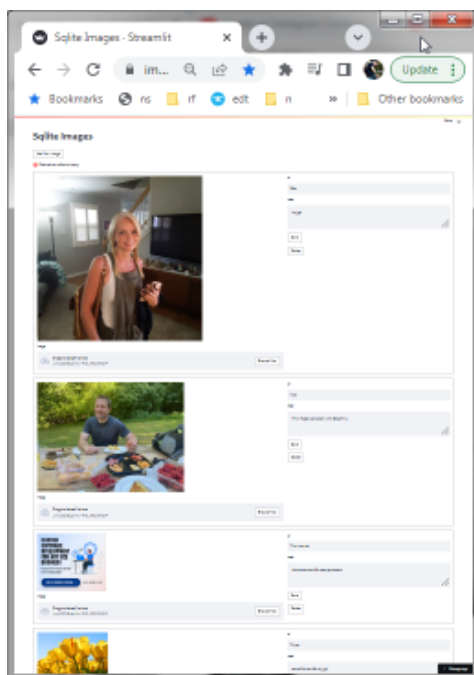
st.title('Sqlite Images')
if st.button('Add New Image'):
    cur.execute('INSERT INTO pics(id, img, note) VALUES(?,?,?)', ('', '', ''))
    con.commit()

for row in cur.execute('SELECT rowid, id, img, note FROM pics ORDER BY id'):
    with st.expander(row[1]):
```

```

with st.form(f'ID-{row[0]}', clear_on_submit=True):
    id=st.text_input('id', row[1])
    if row[2]:
        img=row[2]
        st.image(row[2])
    file=st.file_uploader('Image', ['png', 'jpg', 'gif', 'bmp'])
    if file:
        img=file.read()
    note=st.text_area('note', row[3])
    if st.form_submit_button('Submit'):
        cur.execute(
            'UPDATE pics SET id=?, img=?, note=? WHERE id=?;',
            (id, img, note, str(row[1])))
        )
        con.commit()
        st.experimental_rerun()
    if st.form_submit_button("Delete"):
        cur.execute(f'''DELETE FROM pics WHERE rowid="{row[0]}"''')
        con.commit()
        st.experimental_rerun()

```



El ejemplo anterior es básicamente el mismo programa que en la sección anterior, con algunos widgets cambiados para mostrar y cargar imágenes, así como las siguientes evaluaciones adicionales 'si' para mostrar imágenes, o no, según si una imagen ya se ha cargado:

```

if row[2]:
    img=row[2]
    st.image(row[2])
file=st.file_uploader('Image', ['png', 'jpg', 'gif', 'bmp'])
if file:
    img=file.read()

```

7.6 Trabajar con fechas y horas - Aplicación de base de datos de calendario

Aquí hay una variación más del ejemplo anterior que muestra cómo trabajar con fechas y horas en SQL, cómo formatear esos valores usando la biblioteca de fecha y hora de Python y mostrarlos/interactuar usando los métodos de entrada/visualización de Streamlit:

```

import streamlit as st, sqlite3, datetime
con=sqlite3.connect('calendar.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS cal(date DATE, time TEXT, event TEXT)')

if st.button('Add New Event'):
    cur.execute(

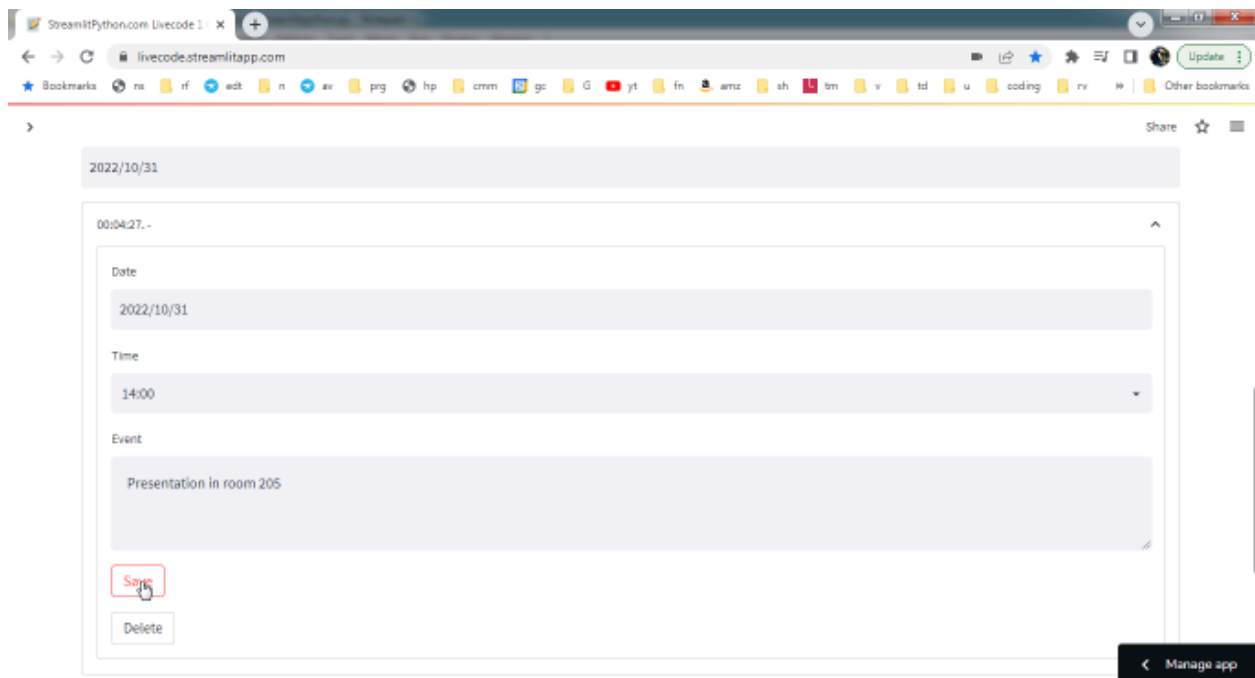
```

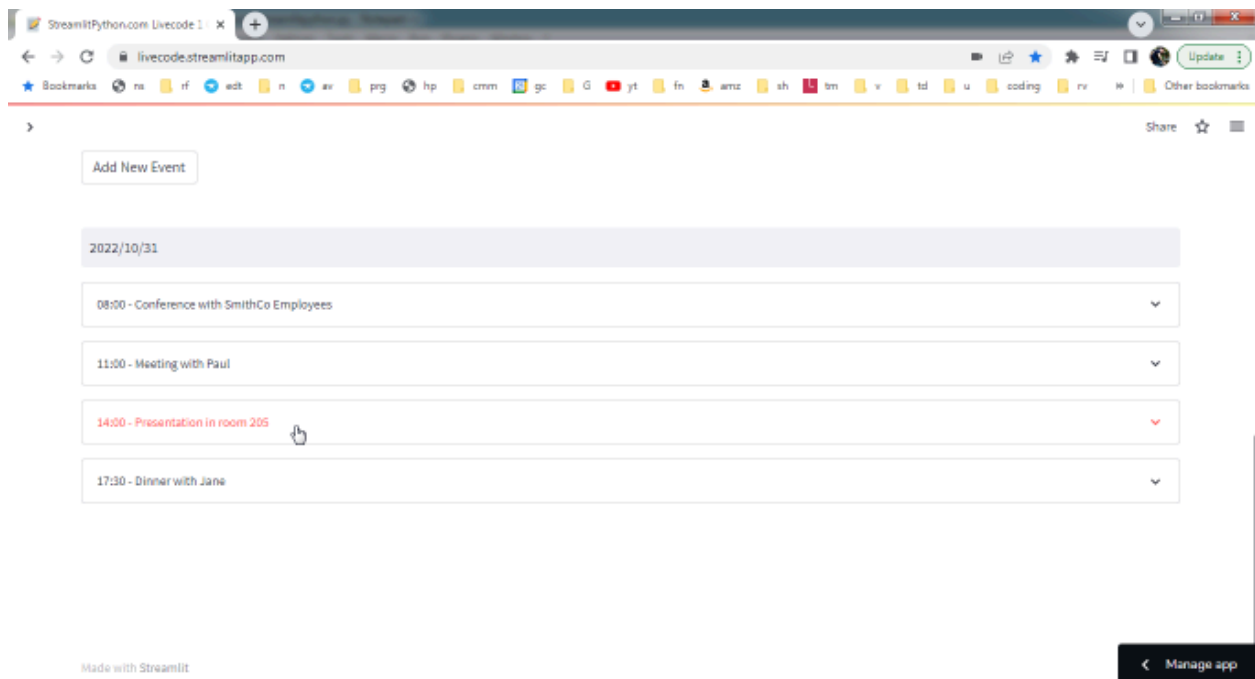
```

'''
    INSERT INTO cal(date, time, event) VALUES(?,?,?)
    ''', (datetime.date.today(), str(datetime.datetime.now().time()), '')
)
con.commit()
st.write('Added to current calendar day/time')

selected_date=st.date_input('Date to View', label_visibility='hidden')
if selected_date:
    for row in cur.execute(
        f'''
            SELECT rowid, date, time, event
            FROM cal
            WHERE date = '{str(selected_date)}'
            ORDER BY time
        '''
    ):
        with st.expander(f'{row[2][:-6]} - {row[3]}'):
            with st.form(f'ID-{row[0]}'):
                date=st.date_input(
                    'Date', datetime.datetime.strptime(row[1], '%Y-%m-%d').date()
                )
                time=st.time_input(
                    'Time', datetime.datetime.strptime(row[2], '%H:%M:%S.%f').time()
                )
                event=st.text_area('Event', row[3])
                if st.form_submit_button('Save'):
                    cur.execute(
                        'UPDATE cal SET date=?, time=?, event=? WHERE date=? and time=?;',
                        (date, str(time)+'.00', event, row[1], row[2])
                    )
                    con.commit() ; st.experimental_rerun()
                if st.form_submit_button("Delete"):
                    cur.execute(f'DELETE FROM cal WHERE rowid="{row[0]}"')
                    con.commit() ; st.experimental_rerun()

```





Tenga en cuenta que en lugar de `ast.literal_eval()`, el código anterior usa métodos de la biblioteca de fecha y hora de Python para convertir datos a/desde valores devueltos por declaraciones SQL:

```
datetime.date.today()
str(datetime.datetime.now().time())
datetime.datetime.strptime(row[1], '%Y-%m-%d').date()
datetime.datetime.strptime(row[1], '%Y-%m-%d').date()
```

Una búsqueda rápida en Google proporcionará un sinnúmero de tutoriales sobre los métodos de fecha y hora de Python, si desea obtener más información.

8. Algunos patrones de diseño crudos más típicos, columnas y formularios individuales

8.1 Pasar de los diseños repetitivos con `St.expander()`, usando `St.columns()`

La técnica `st.expander()` que se muestra en los ejemplos anteriores de la base de datos es útil en muchos casos, pero puede volverse difícil de manejar cuando hay muchas filas para mostrar. Y en el caso de la visualización de imágenes, no es conveniente hacer que el usuario haga clic en un menú desplegable para ver cada imagen individual.

Aquí hay un ejemplo usando `st.columns()` para diseñar imágenes y widgets de formulario en la pantalla. Este tipo de diseño es útil para cualquier tipo de visualización de inventario, o para otros tipos de aplicaciones con diseños repetidos de desplazamiento, donde cada fila de resultados tiene la misma configuración visual de 'tarjeta'. Imagine, por ejemplo, cómo se muestran los resultados de Amazon y Ebay, no necesariamente en cuadrículas similares a hojas de cálculo, sino con un diseño de diseño visual repetido utilizado para mostrar cada registro en un conjunto de resultados:

```
import streamlit as st, sqlite3

con=sqlite3.connect('picscols.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS pics(id TEXT, img BLOB, note TEXT)')

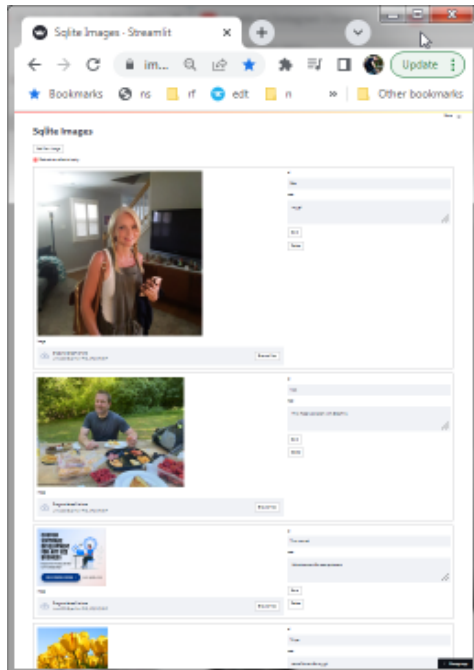
st.title('Sqlite Images')
if st.button('Add New Image'):
    cur.execute('INSERT INTO pics(id, img, note) VALUES(?,?,?)', ('', '', ''))
    con.commit()

for row in cur.execute('SELECT rowid, id, img, note FROM pics ORDER BY id'):
    with st.form(f'ID-{row[0]}', clear_on_submit=True):
        imgcol, notecol = st.columns([3, 2])
        id=notecol.text_input('id', row[1])
        note=notecol.text_area('note', row[3])
        if row[2]:
```

```

img=row[2]
imgcol.image(row[2])
file=imgcol.file_uploader('Image', ['png', 'jpg', 'gif', 'bmp'])
if file:
    img=file.read()
if notecol.form_submit_button('Save'):
    cur.execute(
        'UPDATE pics SET id=?, img=?, note=? WHERE id=?;',
        (id, img, note, str(row[1]))
    )
    con.commit()
    st.experimental_rerun()
if notecol.form_submit_button("Delete"):
    cur.execute(f'''DELETE FROM pics WHERE rowid="{row[0]}"''')
    con.commit()
    st.experimental_rerun()

```



La mayor parte del código anterior es idéntico al ejemplo de la sección anterior, con solo un poco de refactorización para usar diseños de columna. Observe particularmente la línea:

```
imgcol, notecol = st.columns([3, 2])
```

Eso crea 2 columnas de visualización llamadas 'imagecol' y 'notecol', con tamaños horizontales en la proporción 3:2. Entonces, en lugar de usar `st.`, `.` se utiliza para colocar widgets en una columna determinada. Por ejemplo, la siguiente línea coloca un widget `st.text_input()` en la columna 'notecol', mostrando inicialmente el valor de la fila[1] seleccionado de la base de datos, y asigna el valor ingresado por el usuario a la variable 'id':

```
id=notecol.text_input('id', row[1])
```

¡Eso es mucho trabajo logrado con muy poco código!

Cada uno de los diseños repetidos en el ejemplo anterior podría ser tan arbitrariamente complicado como desee. Simplemente incluya cualquier diseño que cree en un bucle 'for', y obtendrá un método maravillosamente simple y componible para entregar diseños repetidos de desplazamiento en aplicaciones de todo tipo.

8.2 Diseños tradicionales de un solo formulario

Hay algunos otros patrones de visualización y diseño que se usan comúnmente en diversas situaciones de aplicación. Uno de los patrones de diseño más comunes es presentar un solo formulario para mostrar / editar detalles de fila, con un widget de lista desplegable que se usa para seleccionar una fila en particular de la base de datos.

En el código siguiente, encontrará las mismas consultas SQL utilizadas en los ejemplos anteriores, pero con un solo formulario completado por los resultados de un solo registro seleccionado de la base de datos:

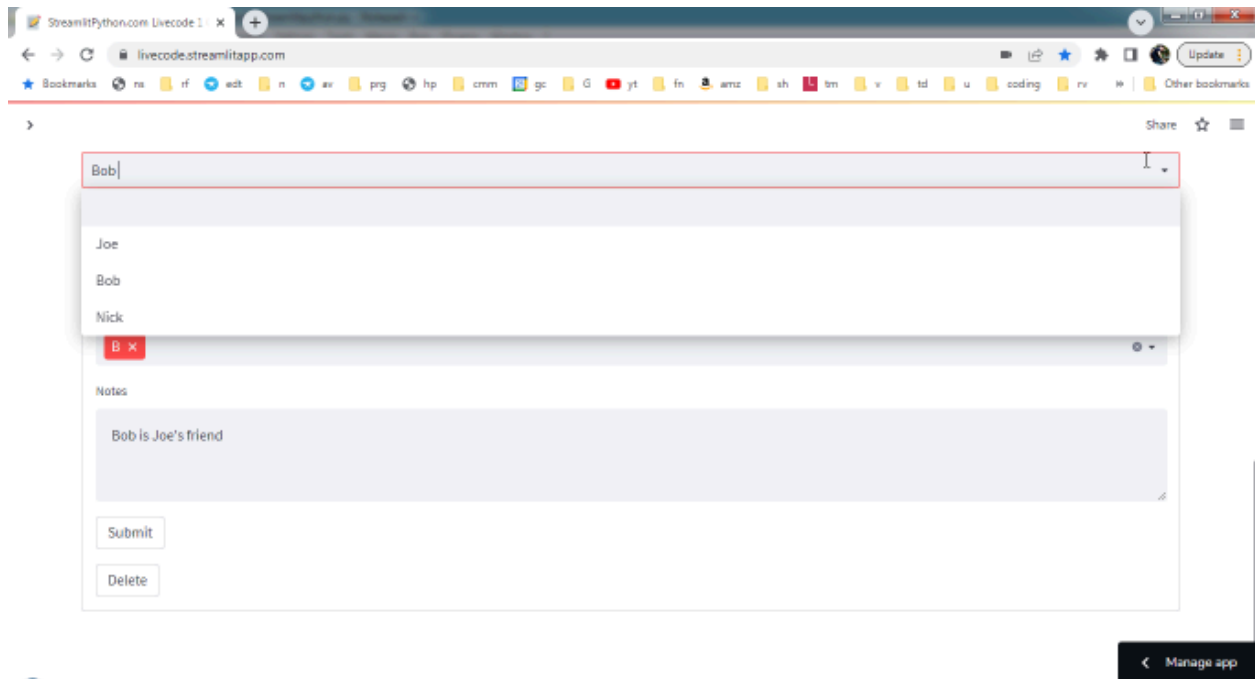
```

import streamlit as st, sqlite3, ast

con=sqlite3.connect('db.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')

st.title('CRUD Single Form')
st.header("Create/Update/Delete")
names=[str(row[0]) for row in cur.execute(f"SELECT name FROM db;")]
names.insert(0, '')
name_to_update=st.selectbox('', names)
if name_to_update!='':
    createorupdate='update'
    row_to_update=cur.execute(
        f'''SELECT * FROM db WHERE name="{name_to_update}";'''
    ).fetchall()[0]
    nameval=row_to_update[0]
    lettersval=ast.literal_eval(row_to_update[1])
    noteval=row_to_update[2]
else:
    createorupdate='create'
    nameval=''
    lettersval=[]
    noteval=''
with st.form("Create or Update a Row of Data", clear_on_submit=True):
    name=st.text_input('Name', value=nameval)
    letters=st.multiselect('Letters', ['A', 'B', 'C'], default=lettersval)
    note=st.text_area('Notes', placeholder='Some Text', value=noteval)
    if st.form_submit_button("Submit"):
        if createorupdate=='create':
            cur.execute(
                '''
                INSERT INTO db(name, letters, note)
                VALUES(?,?,?)
                ''', (name, str(letters), note)
            )
            con.commit()
        else:
            cur.execute(
                '''
                UPDATE db
                SET name=?, letters=?, note=?
                WHERE name=?;
                ''', (name, str(letters), note, str(nameval))
            )
            con.commit()
            st.experimental_rerun()
if st.form_submit_button("Delete"):
    cur.execute(f'''DELETE FROM db WHERE name="{name_to_update}";''')
    con.commit()
    st.write(f"{name_to_update} has been deleted")

```



En este ejemplo, los valores de nombre de cada fila de la base de datos se rellenan en un selectbox, utilizando una comprensión de lista en las filas devueltas por una consulta de selección SQL. Se agrega una cadena vacía a los resultados de la consulta (insertada en el índice 0) para permitir a los usuarios crear un nuevo registro:

```
names=[str(row[0]) for row in cur.execute(f"SELECT name FROM db;")]
names.insert(0, '')
name_to_update=st.selectbox('', names)
```

Tenga en cuenta que el widget selectbox sugiere una breve *lista de selección de autocompletar*, cada vez que un usuario *comienza a escribir un valor parcial*. Esto es muy útil cuando la lista de elementos seleccionables en una lista desplegable es grande, de modo que los usuarios nunca tengan que desplazarse por una larga lista de elementos para encontrar un valor determinado.

El resto del código anterior utiliza instrucciones SQL para crear, actualizar y eliminar registros de base de datos de la misma manera que en los ejemplos anteriores. Se utilizan varias evaluaciones condicionales "if" para determinar si el contenido del formulario debe usarse para crear un nuevo registro o para actualizar el registro mostrado existente:

```
name_to_update=st.selectbox('', names)
if name_to_update!='':
    createorupdate='update'
```

Si el valor seleccionado en la lista desplegable es una cadena vacía, se crea un nuevo registro. De lo contrario, el registro mostrado se actualiza con los cambios introducidos por el usuario (en este ejemplo se utilizan las mismas instrucciones SQL textuales de ejemplos anteriores):

```
if createorupdate=='create':
    cur.execute(
        '''
        INSERT INTO db(name, letters, note)
        VALUES(?,?,?)
        ''', (name, str(letters), note)
    )
    con.commit()
else:
    cur.execute(
        '''
        UPDATE db
        SET name=?, letters=?, note=?
        WHERE name=?;
        ''', (name, str(letters), note, str(nameval))
    )
```



```
con.commit()
st.experimental_rerun()
```

8.3 Uso de cuadrículas de tramas de datos para mostrar el contenido de la base de datos

Podemos crear una vista de cuadrícula compacta de todas las filas de la base de datos devueltas por una consulta SQL, utilizando un widget `st.dataframe()`. `st.dataframe()` está destinado a mostrar marcos de datos de Pandas, que se pueden generar fácilmente a partir de listas devueltas por consultas de bases de datos:

```
import streamlit as st, pandas as pd, sqlite3

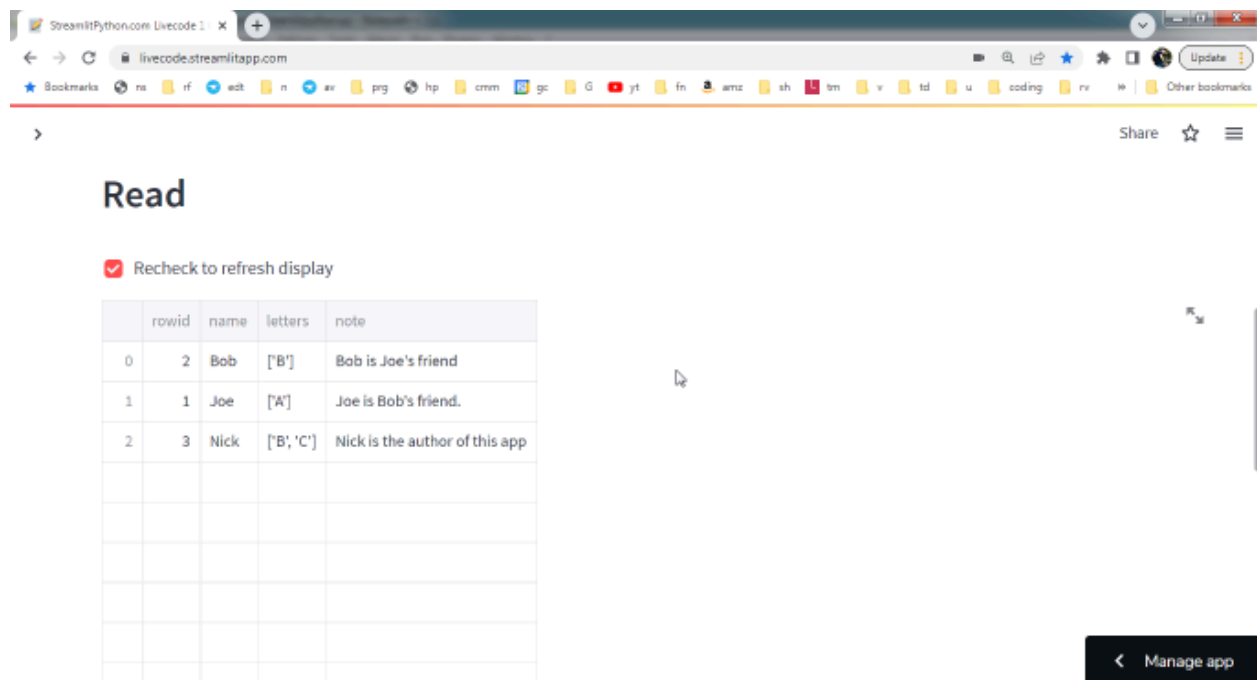
con=sqlite3.connect('db.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')
st.header("Read")

#### The following code assigns the variable 'myrows' to
#### the results of an SQL query:

if st.checkbox('Recheck to refresh display', value=True):
    myrows=[
        list(row) for row in cur.execute(
            '''
                SELECT rowid, name, letters, note
                FROM db
                ORDER BY name
            '''
        )
    ]

#### Here we create a dataframe from the 'myrows' list
#### generated by the SQL query above:

st.dataframe(
    pd.DataFrame(
        myrows,
        columns=['rowid', 'name', 'letters', 'note'],
        height=800
    )
)
```



	rowid	name	letters	note
0	2	Bob	['B']	Bob is Joe's friend
1	1	Joe	['A']	Joe is Bob's friend.
2	3	Nick	['B', 'C']	Nick is the author of this app

Observe el uso de `st.checkbox()` para actualizar el diseño anterior. Cubriremos cómo funciona esto con más detalle, en una sección posterior titulada 'Flujo de programa Streamlit'.

En el ejemplo siguiente se combinan las visualizaciones anteriores de formularios y marcos de datos en un diseño de columnas en paralelo:

```

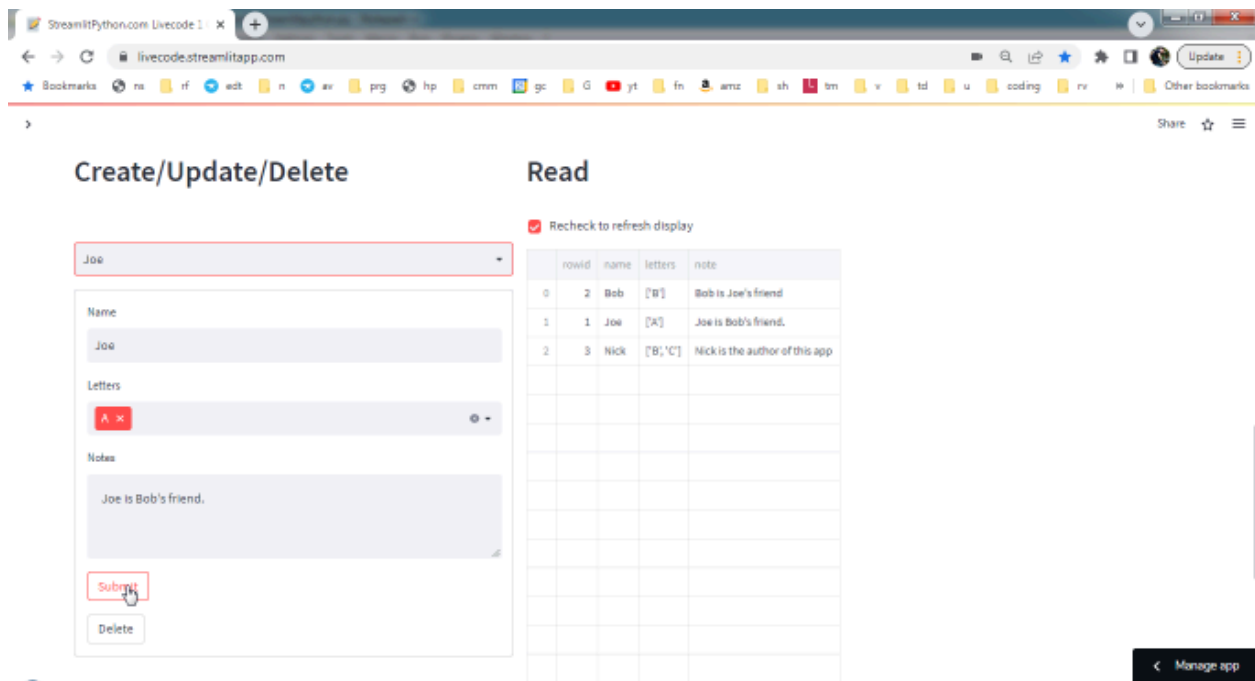
import streamlit as st, pandas as pd, sqlite3, ast

con=sqlite3.connect('db.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')

st.title('CRUD Single Form')
col1, col2=st.columns([2,3])
with col1:
    st.header("Create/Update/Delete")
    names=[str(row[0]) for row in cur.execute(f"SELECT name FROM db;")]
    names.insert(0, '')
    name_to_update=st.selectbox('', names)
    if name_to_update!='':
        createorupdate='update'
        row_to_update=cur.execute(
            f'''SELECT * FROM db WHERE name="{name_to_update}";'''
        ).fetchall()[0]
        nameval=row_to_update[0]
        lettersval=ast.literal_eval(row_to_update[1])
        noteval=row_to_update[2]
    else:
        createorupdate='create'
        nameval=''
        lettersval=[]
        noteval=''
    with st.form("Create or Update a Row of Data", clear_on_submit=True):
        name=st.text_input('Name', value=nameval)
        letters=st.multiselect('Letters', ['A', 'B', 'C'], default=lettersval)
        note=st.text_area('Notes', placeholder='Some Text', value=noteval)
        if st.form_submit_button("Submit"):
            if createorupdate=='create':
                cur.execute(
                    '''
                        INSERT INTO db(name, letters, note)
                        VALUES(?,?,?)
                    ''', (name, str(letters), note)
                )
                con.commit()
            else:
                cur.execute(
                    '''
                        UPDATE db
                        SET name=?, letters=?, note=?
                        WHERE name=?;
                    ''', (name, str(letters), note, str(nameval))
                )
                con.commit()
                st.experimental_rerun()
        if st.form_submit_button("Delete"):
            cur.execute(f'''DELETE FROM db WHERE name="{name_to_update}";''')
            con.commit()
            st.write(f"{name_to_update} has been deleted")

with col2:
    st.header("Read")
    if st.checkbox('Recheck to refresh display', value=True):
        myrows=[
            list(row) for row in cur.execute(
                '''
                    SELECT rowid, name, letters, note
                    FROM db
                    ORDER BY name
                '''
            )
        ]
    st.dataframe(
        pd.DataFrame(
            myrows,
            columns=['rowid', 'name', 'letters', 'note']),
        height=800
    )

```



9. Aplicaciones de ejemplo más simples para demostrar los conceptos básicos de Streamlit en la práctica

Esta sección del tutorial consta de una variedad de ejemplos de aplicaciones simples. Algunos son triviales, pero todos juntos demuestran cómo componer bloques de construcción comunes necesarios para implementar funcionalidades utilizadas en muchos tipos comunes de aplicaciones.

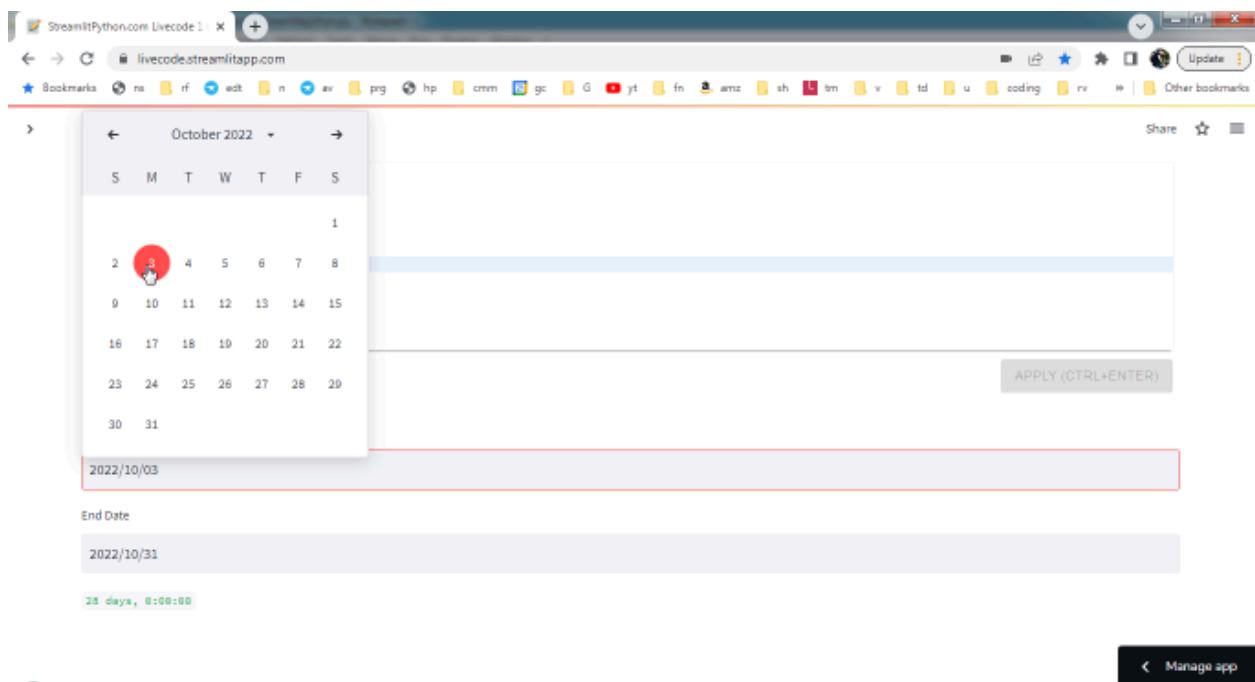
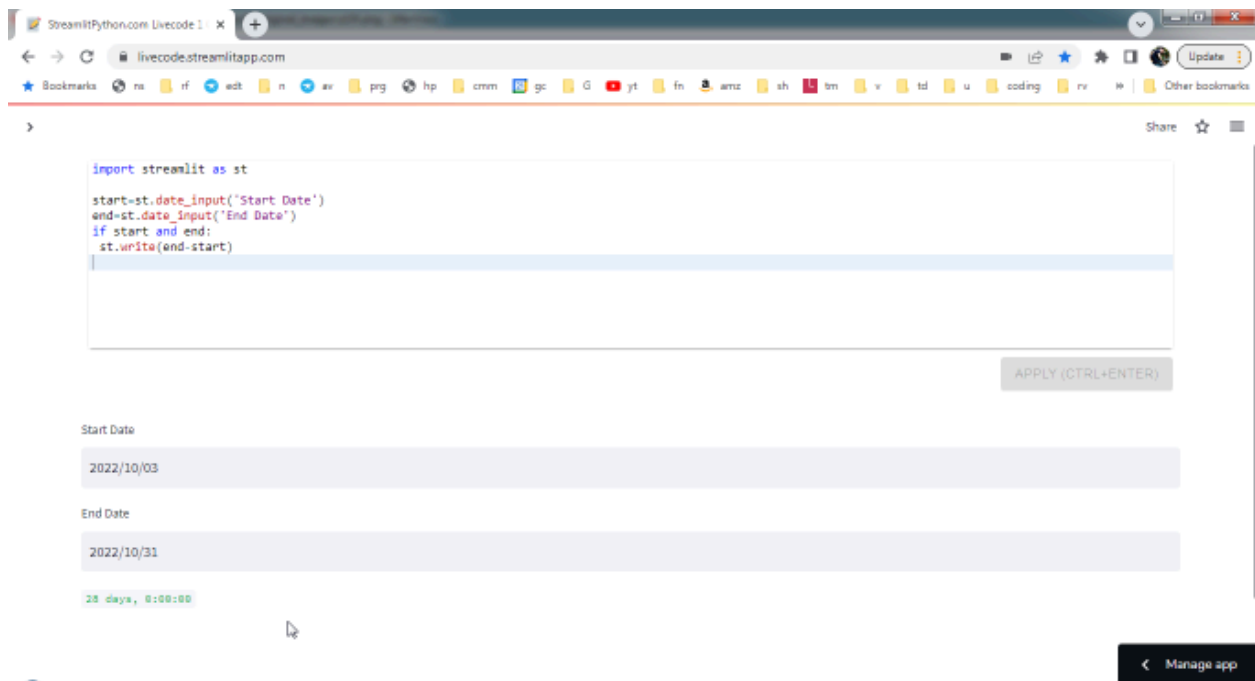
Verá que la mayoría de las aplicaciones de esta sección requieren solo unas pocas líneas de código Streamlit para implementar la entrada del usuario y la salida mostrada. La mayor parte del código es Python genérico, utilizado para realizar cálculos, manipulaciones de listas, evaluaciones condicionales, bucles, operaciones CRUD, etc.

9.1 Tiempo entre fechas

En este ejemplo se calcula el número de días entre 2 fechas especificadas por el usuario. Utiliza dos widgets `st.date_input()` y muestra los resultados calculados con `st.write()`:

```
import streamlit as st

start=st.date_input('Start Date')
end=st.date_input('End Date')
if start and end:
    st.write(end-start)
```



9.2 Visor de cámara web en vivo

Esta aplicación permite al usuario ver imágenes en vivo de varias cámaras en línea. `st.selectbox()` y `st.image()` son los únicos widgets necesarios:

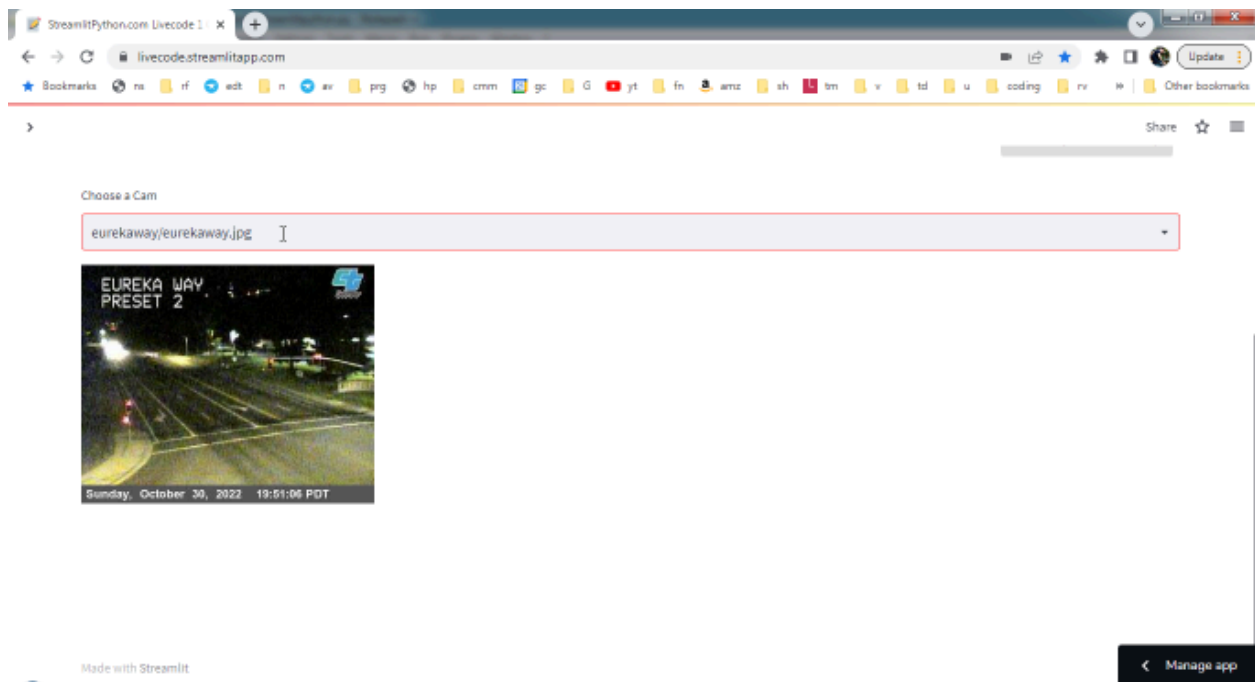
```

import streamlit as st

cam=st.selectbox(
    'Choose a Cam',
    [
        '',
        'pitriverbridge/pitriverbridge.jpg',
        'johnsongrade/johnsongrade.jpg',
        'perez/perez.jpg',
        'mthebron/mthebron.jpg',
        'eurekaway/eurekaway.jpg',
        'sr70us395/sr70us395.jpg',
        'bogard/bogard.jpg',
        'eastriverside/eastriverside.jpg',
    ]
)

```

```
if cam:
    st.image('https://cwwp2.dot.ca.gov/data/d2/cctv/image/' + cam)
```

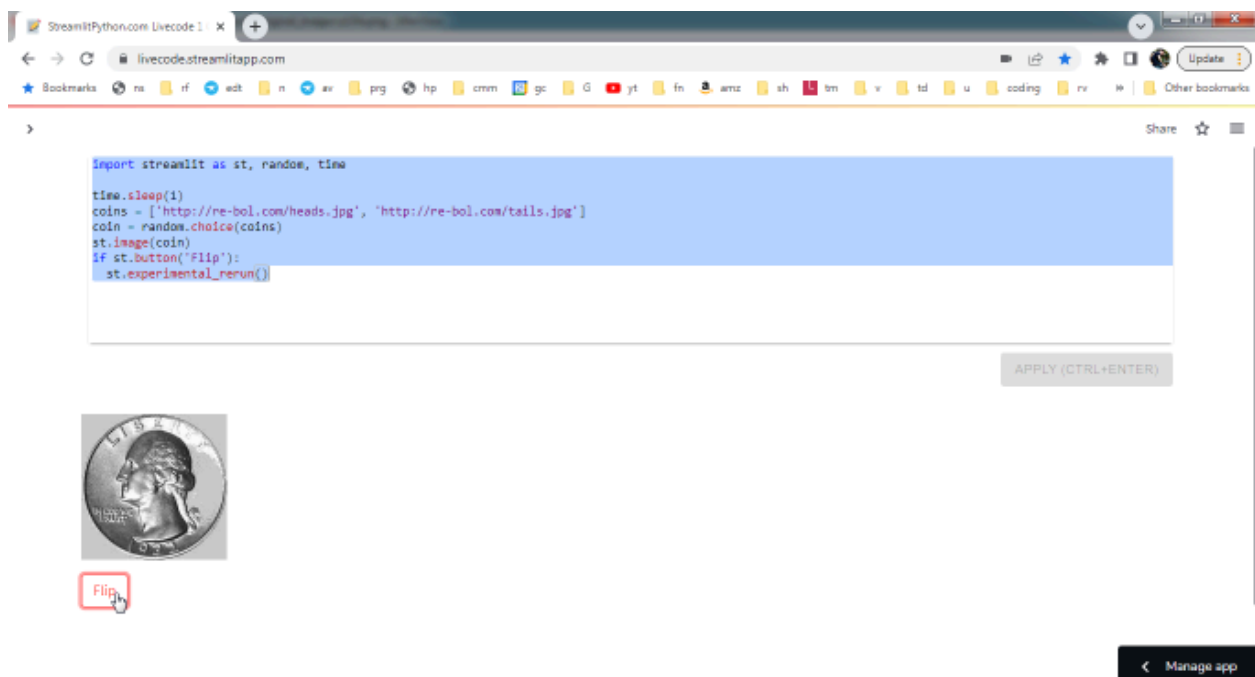


9.3 Lanzamiento de moneda

Esta aplicación utiliza el método `st.experimental_rerun()` para volver a cargar la página cada vez que se hace clic en el botón:

```
import streamlit as st, random, time

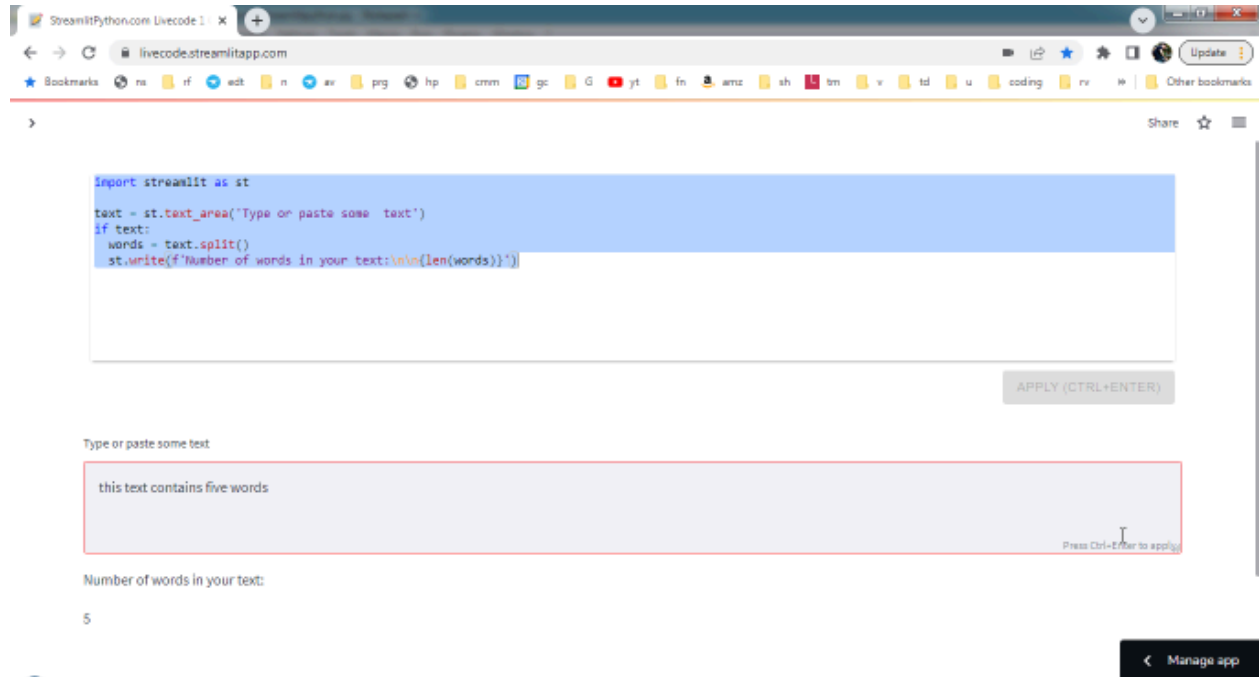
time.sleep(1)
coins = ['http://re-bol.com/heads.jpg', 'http://re-bol.com/tails.jpg']
coin = random.choice(coins)
st.image(coin)
if st.button('Flip'):
    st.experimental_rerun()
```



9.4 Recuento de palabras

Este ejemplo simplemente divide el texto ingresado en un widget `st.text_area()` y muestra un recuento de los valores en esa lista, usando el widget `st.write()`:

```
import streamlit as st
text = st.text_area('Type or paste some text')
if text:
    words = text.split()
    st.write(f'Number of words in your text:\n\n{len(words)}')
```

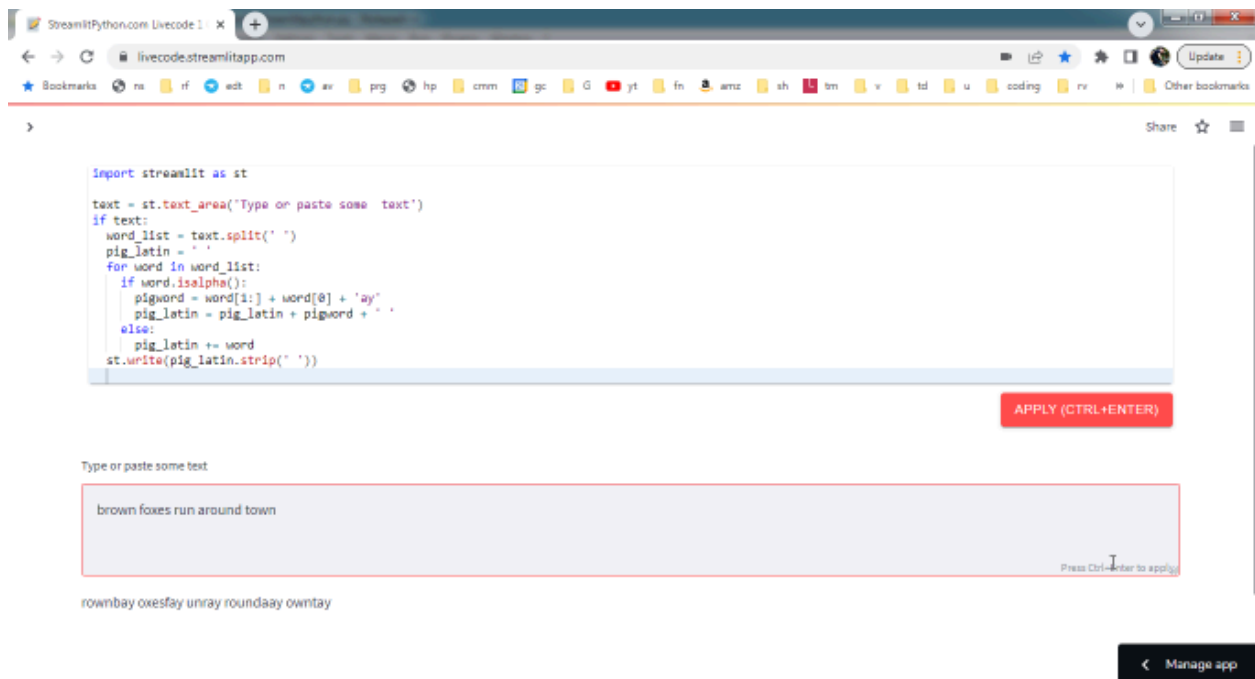


9.5 Generador de latín de cerdo

Nuevamente, los únicos widgets de Streamlit necesarios aquí son `st.text_area()` y `st.write()`. El resto es solo Python genérico que divide y recompone caracteres a partir de la cadena de palabras ingresadas por el usuario:

```
import streamlit as st

text = st.text_area('Type or paste some text')
if text:
    word_list = text.split(' ')
    pig_latin = ' '
    for word in word_list:
        if word.isalpha():
            pigword = word[1:] + word[0] + 'ay'
            pig_latin = pig_latin + pigword + ' '
        else:
            pig_latin += word
    st.write(pig_latin.strip(' '))
```



9.6 Calculadora

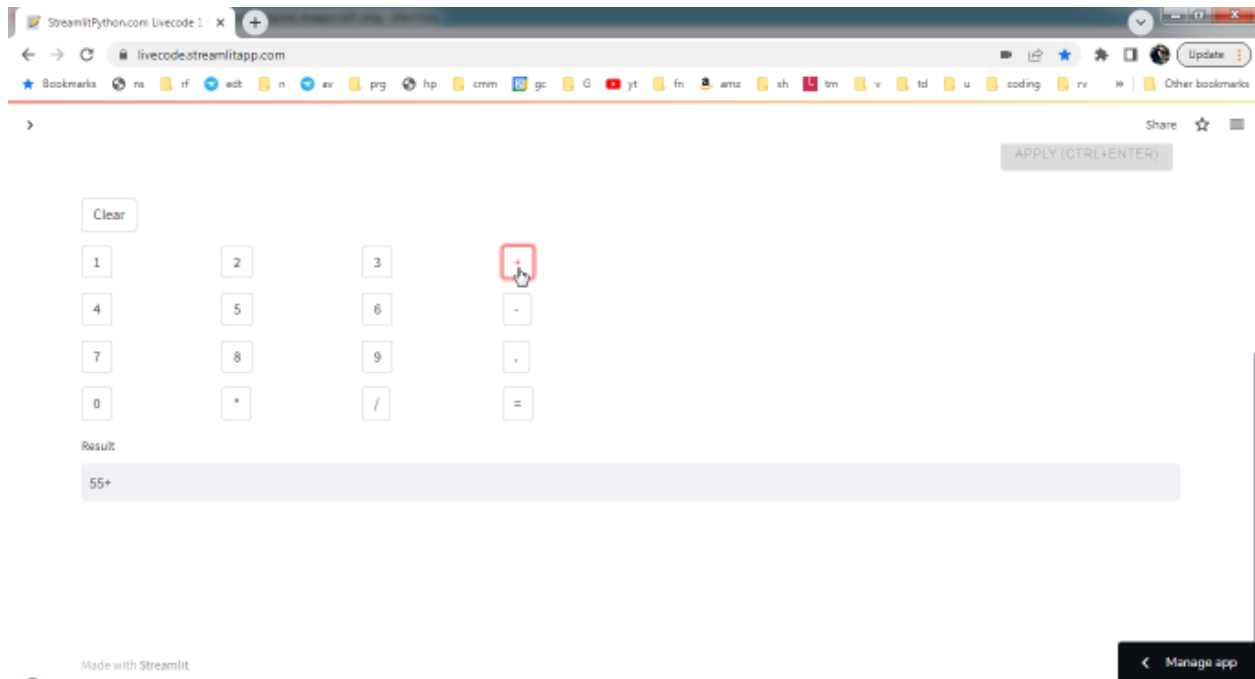
Aquí hay una calculadora GUI de juguete construida con widgets `st.button()`. Su único propósito es demostrar un poco más sobre el diseño visual y el uso de columnas, así como el flujo del programa y el uso de `st.session_state()` para almacenar valores de variables:

```

import streamlit as st, ast

if 'total' not in st.session_state:
    st.session_state.total = ''
if st.button('Clear'): st.session_state.total = ''
col1, col2, col3, col4, col5 = st.columns([1, 1, 1, 1, 4])
if col1.button('1'): st.session_state.total += '1'
if col2.button('2'): st.session_state.total += '2'
if col3.button('3'): st.session_state.total += '3'
if col4.button('+'): st.session_state.total += '+'
if col1.button('4'): st.session_state.total += '4'
if col2.button('5'): st.session_state.total += '5'
if col3.button('6'): st.session_state.total += '6'
if col4.button('-'): st.session_state.total += '-'
if col1.button('7'): st.session_state.total += '7'
if col2.button('8'): st.session_state.total += '8'
if col3.button('9'): st.session_state.total += '9'
if col4.button('.'): st.session_state.total += '.'
if col1.button('0'): st.session_state.total += '0'
if col2.button('*'): st.session_state.total += '*'
if col3.button('/'): st.session_state.total += '/'
if col4.button('='):
    st.session_state.total = str(eval(st.session_state.total))
st.text_input('Result', st.session_state.total)

```

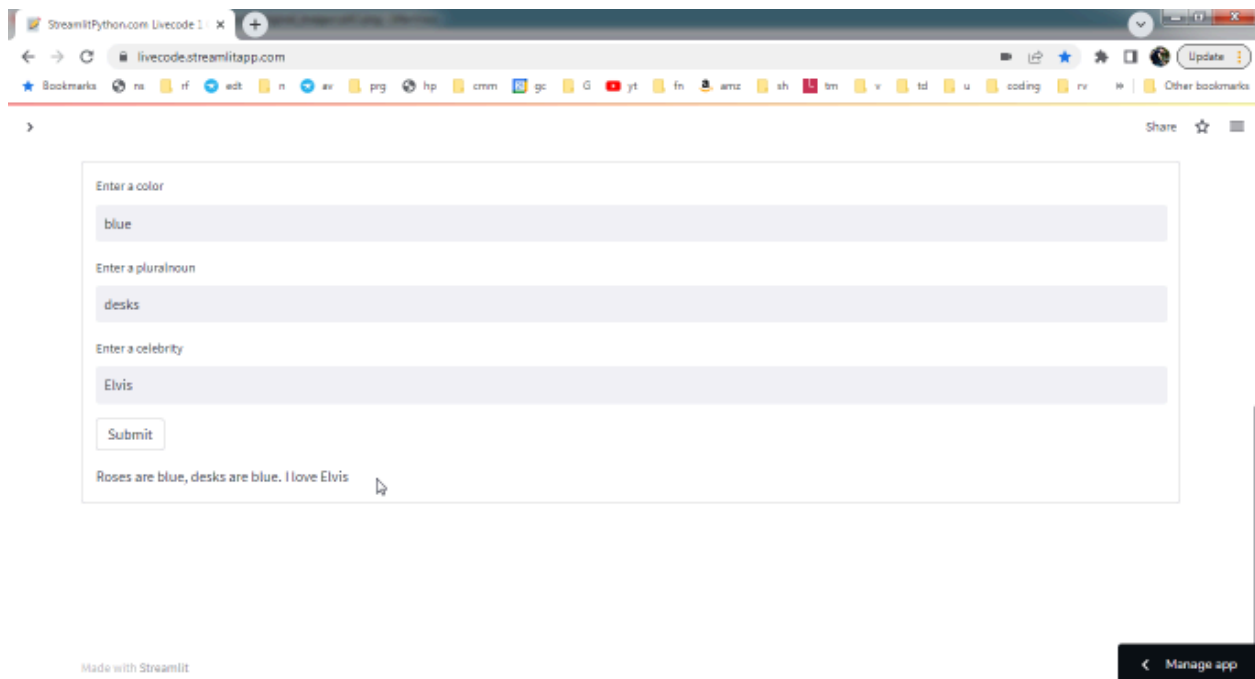


9.7 Mad Libs generados dinámicamente

Con esta aplicación, puede ver el widget `st.checkbox()`, que se usa aquí como algo similar a un `st.button()`, pero que guarda su estado seleccionado en cada reejecución (True, siempre que la casilla de verificación esté marcada). Este ejemplo también demuestra cómo *generar dinámicamente* un número variable de widgets de entrada de formulario, utilizando un bucle 'for', basado en los valores numéricos necesarios para mostrar un madlib dado:

```
import streamlit as st, random

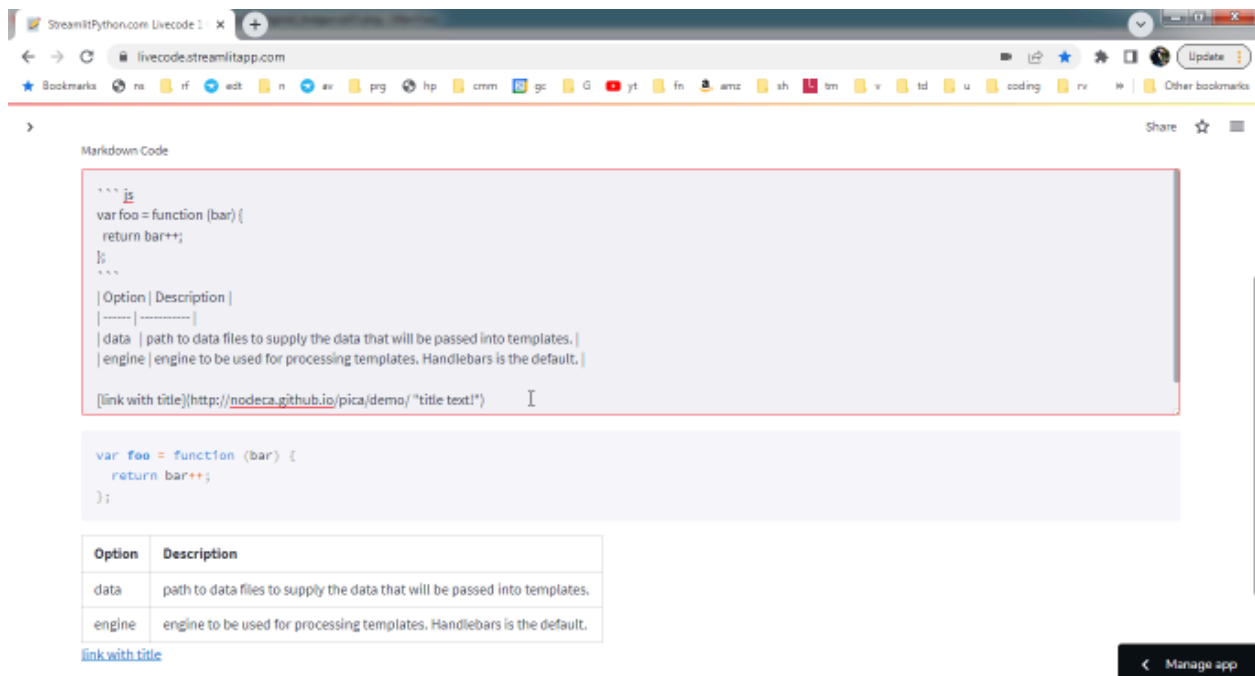
if 'madlibText' not in st.session_state:
    st.session_state.madlibText = ''
st.write(st.session_state.madlibText)
allLines = [
    "person,place,time#I went to the %2 with %1 at %3",
    "object,person#A %1 flew into %2's face.",
    "color,pluralnoun,celebrity#Roses are %1, %2 are blue. I love %3"
]
madlib = random.choice(allLines)
halves = madlib.split("#")
madlibText = halves[1]
inputs = halves[0].split(",")
inputStore = [""]
with st.form("id"):
    widgets=''
    for item in inputs:
        widgets+=f'{item}=st.text_input("Enter a {item}") ; '
    exec(widgets)
    if st.form_submit_button("Submit"):
        code=''
        for item in inputs:
            code+=f'inputStore.append({item})\n'
        exec(code)
        for i in range(len(inputStore)):
            madlibText = madlibText.replace("%"+str(i), inputStore[i])
            madlibText = madlibText.strip()
        st.session_state.madlibText = madlibText
        st.write(st.session_state.madlibText)
```

9.8 Editor de Markdown

Aquí hay un editor simple que permite al usuario ingresar texto con etiquetas de rebajas en un widget `st.text_area()`, y muestra el resultado visual usando un widget `st.markdown()`:

```
import streamlit as st
code=st.text_area('Markdown Code')
if code:
    st.markdown(code)
```



Eso fue demasiado fácil. Agreguemos un código que permita al usuario editar y guardar cualquier número de documentos de Markdown en una base de datos. Esta es solo una ligera variación del ejemplo de base de datos CRUD de forma repetitiva que ha visto anteriormente, con la adición de un widget de rebajas para renderizar los documentos seleccionados:

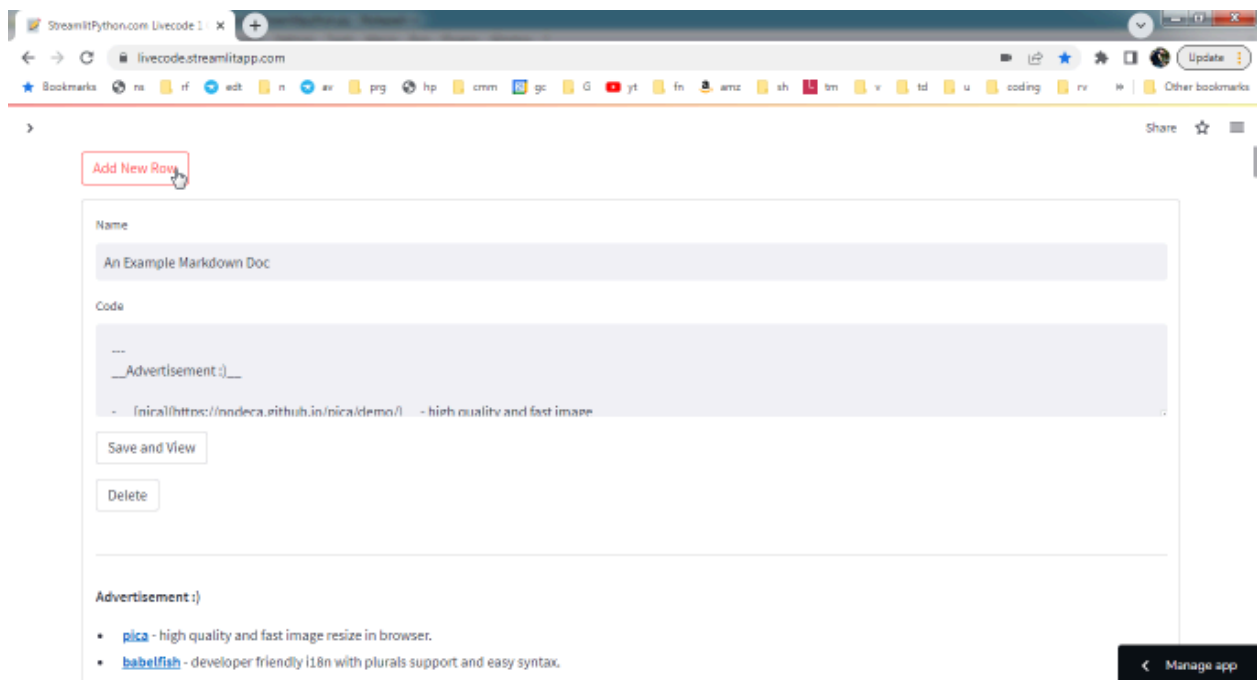
```
import streamlit as st, sqlite3
con=sqlite3.connect('markdown.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, code TEXT)')
```

```

if st.button('Add New Row'):
    cur.execute('INSERT INTO db(name, code) VALUES(?,?)', ('', ''))
    con.commit()

for row in cur.execute('SELECT rowid, name, code FROM db ORDER BY name'):
    with st.form(f'ID-{row[0]}'):
        name=st.text_input('Name', row[1])
        code=st.text_area('Code', row[2])
        if st.form_submit_button('Save and View'):
            cur.execute(
                'UPDATE db SET name=?, code=? WHERE rowid=?;',
                (name, code, str(row[0])))
            con.commit() ; st.experimental_rerun()
        if st.form_submit_button("Delete"):
            cur.execute(f'DELETE FROM db WHERE rowid="{row[0]}"')
            con.commit() ; st.experimental_rerun()
    st.markdown(code)

```



9.9 Sala de chat (con datos de mensajes guardados en un archivo de texto)

Aquí hay una aplicación de sala de chat trivial y básica. Los usuarios escriben su nombre y un mensaje, y esa información se publica para que la lean y respondan otros usuarios que ejecuten la aplicación. También se agrega una marca de tiempo a cada publicación.

Tenga en cuenta que en este ejemplo, las publicaciones de mensajes se escriben en un solo archivo, por lo que si algún usuario intentara escribir en el archivo exactamente en el mismo instante, existe la posibilidad de que el archivo se corrompa. Por lo tanto, este ejemplo solo pretende demostrar algunos patrones de interfaz Streamlit simples, no para uso en producción. Dos ejemplos de tutoriales que siguen a esto se basarán en el diseño básico aquí, pero guardarán los mensajes en una base de datos, de modo que sea más adecuado para el uso en producción.

Observe el argumento 'label_visibility' utilizado en el método `st.text_area()`, que oculta la etiqueta predeterminada sobre el widget de texto. Además de eso, algunos métodos `st.column()`, `st.text_input()`, `st.form()`, `st.form_submit_button()` y `st.experimental_rerun()` son todo el código Streamlit necesario para que esta aplicación funcione:

```

import streamlit as st, datetime
# import os ; os.remove('chat.txt')
# Uncomment the line above to delete the file containing
# all messages, and start over with a fresh chat room.

col1, col2=st.columns([2,3])
with col2:
    with open('chat.txt', 'a+') as file: pass
    with open('chat.txt', 'r+') as file:

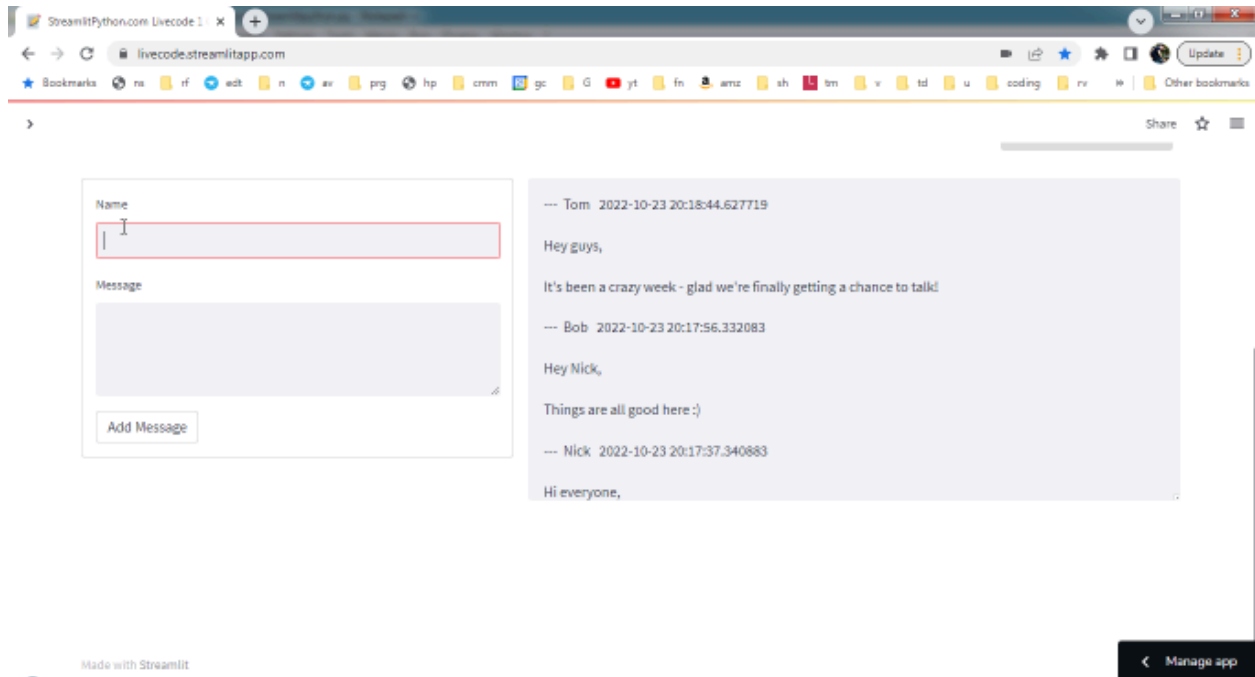
```

```

msg=file.read()
st.text_area('msg', msg, height=350, label_visibility='collapsed')

with col1:
    with st.form('New Message', clear_on_submit=True):
        name=st.text_input('Name')
        message=st.text_area('Message')
        timestamp=datetime.datetime.now()
        if st.form_submit_button('Add Message'):
            newmsg=f'--- {name} {timestamp}\n\n{message}\n\n{msg}'
            with open('chat.txt', 'w') as file:
                file.write(newmsg)
            st.experimental_rerun()

```



9.10 Sala de chat (con datos de mensajes guardados en la base de datos sqlite)

Este ejemplo se basa en la estructura básica del ejemplo de chat anterior, pero guarda los datos en un solo campo de una tabla de base de datos, de modo que sea más adecuado para el uso de producción. Para entornos de producción reales, las aplicaciones que se basan en estos modelos simples se migran mejor de SQLite a cualquier sistema de base de datos multiusuario como PostgreSQL, MySQL, MSSQL, etc.

La mayoría de los comandos SQL de esta versión de la aplicación de chat deberían estar familiarizados con los ejemplos anteriores.

Fíjate en el final comentado de la primera línea. Simplemente descomente esa parte de la línea y vuelva a ejecutar la aplicación, para eliminar completamente la base de datos y comenzar de nuevo (eso hará que todos los mensajes guardados se borren permanentemente):

```

import streamlit as st, sqlite3, datetime#, os ; os.remove('chats.db')
con=sqlite3.connect('chats.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(messages TEXT)')

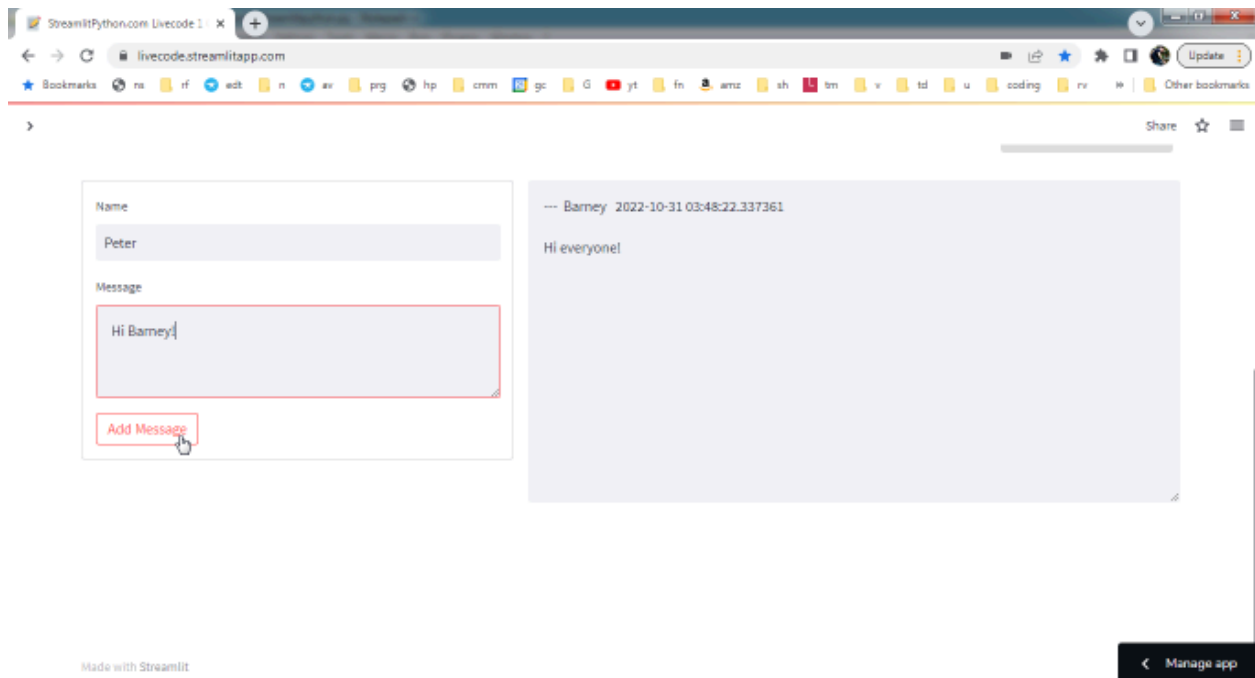
col1, col2=st.columns([2,3])
with col2:
    msg=cur.execute('SELECT messages FROM db').fetchone()
    if msg==None:
        cur.execute('INSERT INTO db(messages) VALUES ("")')
        con.commit()
    st.text_area('msg', msg[0], height=350, label_visibility='collapsed')
with col1:
    with st.form('New Message', clear_on_submit=True):
        name=st.text_input('Name')
        message=st.text_area('Message')
        timestamp=datetime.datetime.now()
        if st.form_submit_button('Add Message'):
            newmsg=f'--- {name} {timestamp}\n\n{message}\n\n{msg[0]}'

```

```

cur.execute(
    'UPDATE db SET messages=? WHERE rowid=?;',
    (newmsg, 1)
)
con.commit()
st.experimental_rerun()

```



9.11 Chat de temas múltiples (multisala)

Este ejemplo se basa más en los dos ejemplos anteriores, para expandir el ejemplo de chat para que los usuarios puedan comunicarse dentro de múltiples hilos de chat, todo en la misma interfaz (muy parecido a un simple foro web):

```

import streamlit as st, sqlite3, datetime #, os ; os.remove('forum.db')
con=sqlite3.connect('forum.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(topic TEXT, messages TEXT)')

# st.set_page_config(layout="wide", page_title="Forum")

for row in cur.execute('SELECT rowid, topic, messages FROM db'):
    st.subheader(row[1])
    with st.expander(''):
        col1, col2=st.columns([3,2])
        with col1:
            st.text_area('topic', row[2], height=350, label_visibility='collapsed')
        with col2:
            with st.form(f'ID{row[0]}', clear_on_submit=True):
                name=st.text_input('Name')
                timestamp=datetime.datetime.now()
                message=st.text_area('Message')
                if st.form_submit_button('Submit'):
                    messages=f'{name} - {timestamp}:\n\n{message}\n\n{row[2]}\n\n'
                    cur.execute(
                        'UPDATE db SET topic=?, messages=? WHERE rowid=?;',
                        (row[1], messages, str(row[0]))
                    )
                    con.commit() ; st.experimental_rerun()

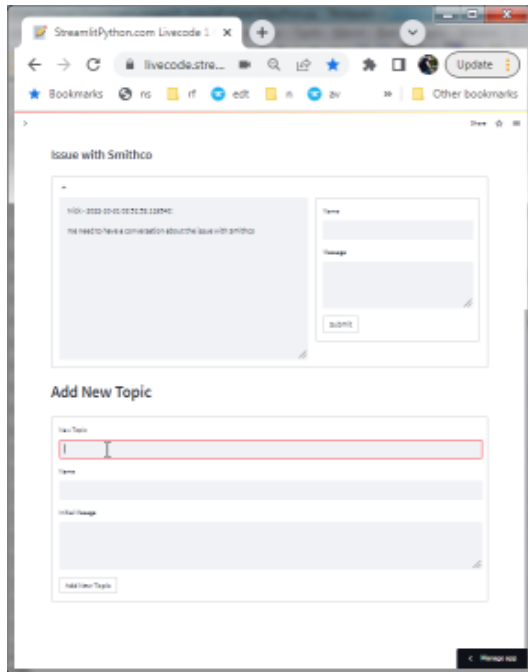
st.header('Add New Topic')
with st.form('New Topic', clear_on_submit=True):
    topic=st.text_input('New Topic')
    name=st.text_input('Name')
    messages=st.text_area('Initial Message')
    timestamp=datetime.datetime.now()
    if st.form_submit_button('Add New Topic'):
        cur.execute('INSERT INTO db(topic, messages) VALUES(?,?)', (

```

```

        topic, f'{name} - {timestamp}:\n\n{messages}\n\n'
    ))
    con.commit() ; st.experimental_rerun()

```



9.12 Caja registradora mínima

Este es un ejemplo de prototipo básico más pequeño posible de una aplicación de pago de caja registradora minorista. Esta primera versión no guarda ningún recibo. Es solo un diseño de pantalla simple que muestra los artículos ingresados para la compra y calcula los cálculos de subtotales, impuestos y ventas totales.

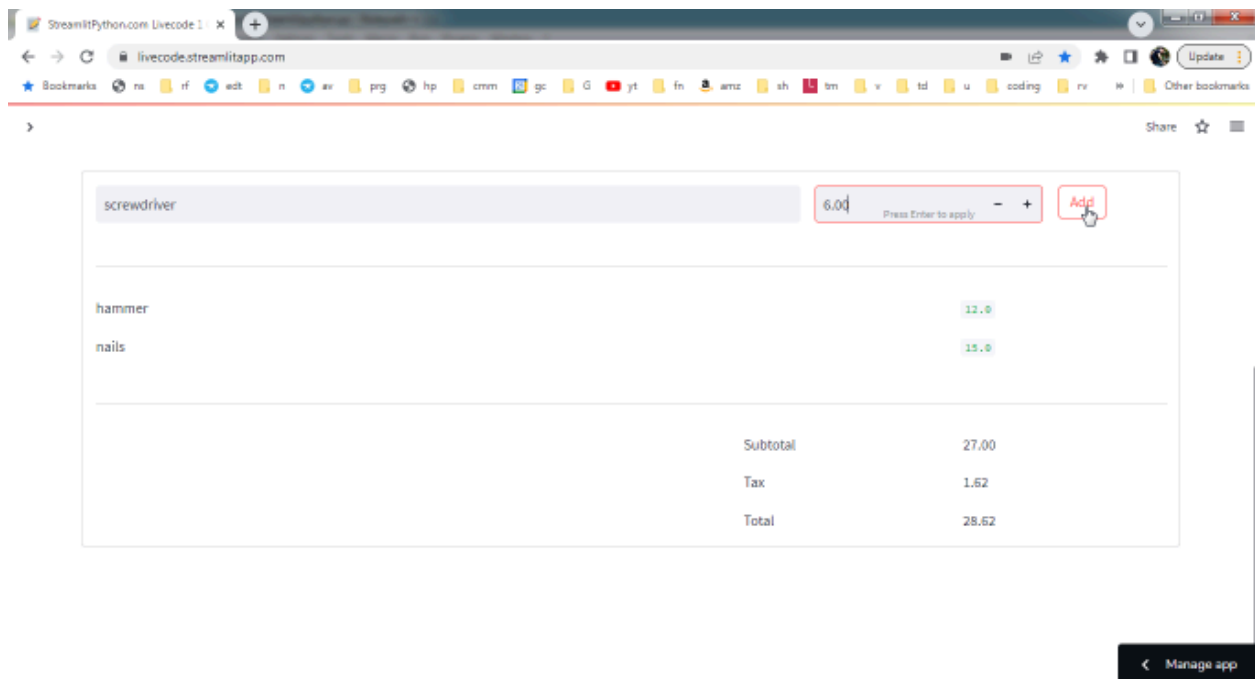
Tenga en cuenta que esta aplicación, y todos los demás ejemplos en el texto, están destinados solo para demostrar patrones de código Streamlit, *no para uso de producción*:

```

import streamlit as st, datetime

if 'purchased' not in st.session_state:
    st.session_state.purchased=[]
with st.form('Add Item', clear_on_submit=True):
    itmcol, prccol, btncol=st.columns([6,2,1])
    item=itmcol.text_input('Item', placeholder='item', label_visibility='collapsed')
    price=prccol.number_input('Price', label_visibility='collapsed')
    if btncol.form_submit_button('Add'):
        timestamp=datetime.datetime.now()
        st.session_state.purchased.append({'item': item, 'price': price})
        st.markdown('---')
        itemcol, pricecol=st.columns([4,1])
        subtotal=0
        for itm in st.session_state.purchased:
            itemcol.write(itm['item'])
            pricecol.write(itm['price'])
            subtotal += (itm['price'])
        st.markdown('---')
        plccol, lblcol, numcol=st.columns([3,1,1])
        lblcol.write('Subtotal')
        numcol.write('{:.2f}'.format(subtotal))
        lblcol.write('Tax')
        numcol.write('{:.2f}'.format(subtotal* .06))
        lblcol.write('Total')
        numcol.write('{:.2f}'.format(subtotal * 1.06))

```



9.13 Caja registradora Segunda parte: Guardar recibos y generar informes de ventas

Para agregar la capacidad de guardar recibos generados por la aplicación de caja registradora, agregaremos algunos comandos SQL familiares que guardan y recuperan los artículos vendidos durante cada transacción de venta, junto con una marca de tiempo cuando se guardó cada recibo. Guardaremos la lista de artículos vendidos en cada recibo y la marca de tiempo, ambos como cadenas en una tabla de base de datos SQLite, como ha visto en ejemplos anteriores.

Para generar informes de ventas, usaremos `ast.literal_eval()` para deserializar la estructura de la lista que contiene los artículos vendidos en cada recibo de ventas. Usaremos la biblioteca de fecha y hora para evaluar los rangos de fechas de inicio y finalización de cada informe, y simplemente sumaremos todos los artículos vendidos en cada recibo seleccionado, para generar el cálculo total de ventas:

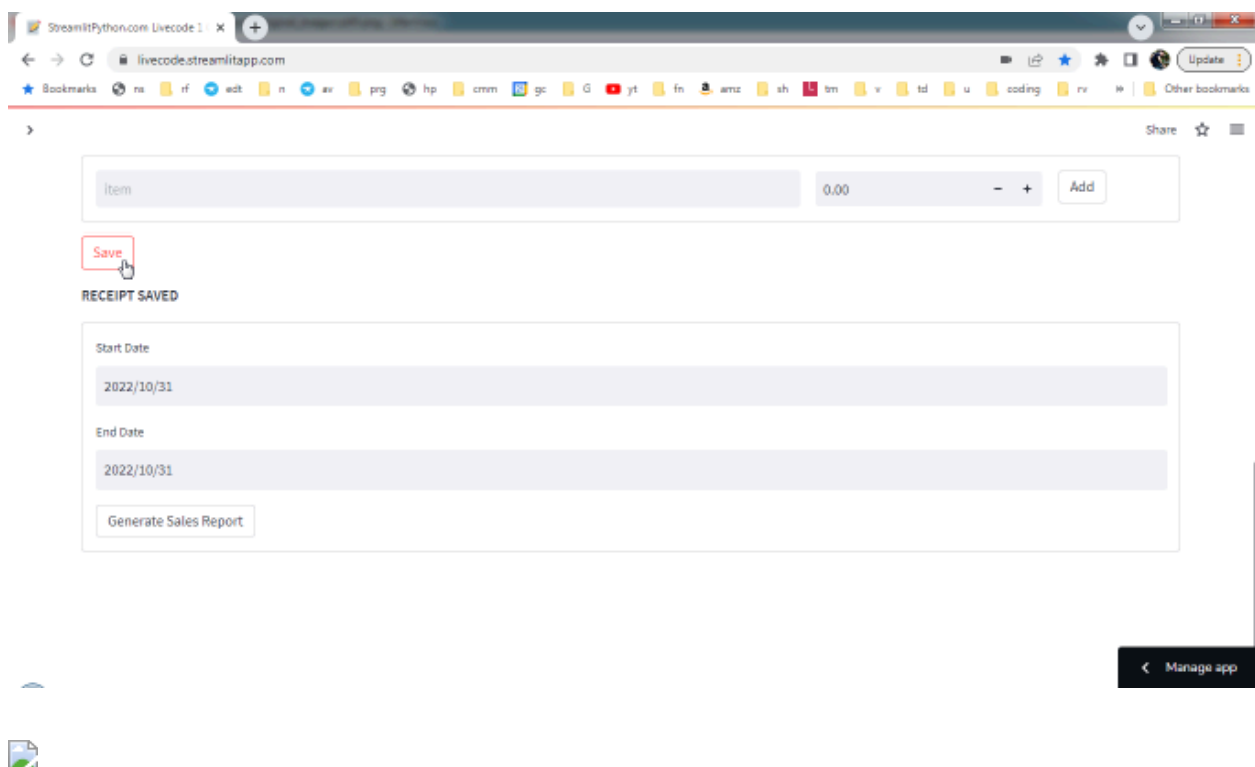
```
import streamlit as st, datetime, sqlite3, ast#, os ; os.remove('sales.db')

con=sqlite3.connect('sales.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS sales(items TEXT, timestamp TEXT)')
if 'purchased' not in st.session_state:
    st.session_state.purchased=[]
with st.form('Add Item', clear_on_submit=True):
    itmcol, prccol, btncol=st.columns([6,2,1])
    item=itmcol.text_input('Item', placeholder='item', label_visibility='collapsed')
    price=prccol.number_input('Price', label_visibility='collapsed')
    if btncol.form_submit_button('Add'):
        timestamp=datetime.datetime.now()
        st.session_state.purchased.append({'item': item, 'price': price})
        st.markdown('---')
        itemcol, pricecol=st.columns([4,1])
        subtotal=0
        for itm in st.session_state.purchased:
            itemcol.write(itm['item'])
            pricecol.write(itm['price'])
            subtotal += (itm['price'])
        st.markdown('---')
        plccol, lblcol, numcol=st.columns([3,1,1])
        lblcol.write('Subtotal')
        numcol.write('{:.2f}'.format(subtotal))
        lblcol.write('Tax')
        numcol.write('{:.2f}'.format(subtotal* .06))
        lblcol.write('Total')
        numcol.write('{:.2f}'.format(subtotal * 1.06))
if st.button('Save'):
    if st.session_state.purchased==[]:
        st.markdown('***Nothing to save on this receipt**')
    else:
        cur.execute('INSERT INTO sales(items, timestamp) VALUES(?,?)', (
            str(st.session_state.purchased), str(datetime.datetime.now())
```

```

))
con.commit()
st.session_state.purchased=[]
st.markdown('**RECEIPT SAVED**')
with st.form('Report', clear_on_submit=True):
    date1=st.date_input('Start Date')
    date2=st.date_input('End Date')
    if st.form_submit_button('Generate Sales Report'):
        sales_total = 0
        for row in cur.execute('''
            SELECT rowid, items, timestamp FROM sales ORDER BY timestamp
        '''):
            saledate=datetime.datetime.strptime(row[2], '%Y-%m-%d %H:%M:%S.%f').date()
            if (
                saledate >= date1 and
                saledate <= date2
            ):
                for item in ast.literal_eval(row[1]):
                    sales_total += item['price']
        st.write(sales_total)

```



9.14 Reloj de turno

Esta aplicación proporciona un botón para que los usuarios marquen la entrada y los turnos de trabajo. Tenga en cuenta que esta aplicación, y todos los demás ejemplos en el texto, están destinados solo para demostrar patrones de código de Streamlit, *no para uso de producción*.

```

import streamlit as st, ast, datetime, sqlite3
con=sqlite3.connect('shifts.db')
cur=con.cursor()
cur.executescript('''
    CREATE TABLE IF NOT EXISTS employees(name TEXT);
    CREATE TABLE IF NOT EXISTS shifts(name TEXT, login TEXT, logout TEXT);
''')
con.commit()

st.header('Shift Clock')
names=[str(row[0]) for row in cur.execute('SELECT name FROM employees')]
selected_name=st.selectbox('Pick Employee', names)

if st.button('Log In'):
    cur.execute(
        'INSERT INTO shifts(name,login) VALUES(?,?)',
        (selected_name, str(datetime.datetime.now()))
    )

```

```

    )
    con.commit()
    st.write('Logged In')

if st.button('Log Out'):
    cur.execute(
        'INSERT INTO shifts(name,logout) VALUES(?,?)',
        (selected_name, str(datetime.datetime.now()))
    )
    con.commit()
    st.write('Logged Out')

st.subheader('Calculate Hours for Selected Employee')
date1=st.date_input('Start Date')
date2=st.date_input('End Date')
if st.button('Calculate Hours') and date1 and date2:
    total_hours = 0
    startdate = 0
    logins = []
    logouts = []
    for row in cur.execute('SELECT name, login, logout FROM shifts'):
        if row[1] is not None:
            login=datetime.datetime.strptime(row[1], '%Y-%m-%d %H:%M:%S.%f')
        else:
            login=None
        if row[2] is not None:
            logout=datetime.datetime.strptime(row[2], '%Y-%m-%d %H:%M:%S.%f')
        else:
            logout=None
        if row[0]==selected_name:
            if logout is None:
                if login >= datetime.datetime.combine(date1, datetime.time(0, 0)):
                    logins.append(login)
                    startdate = login
            else:
                if logout <= datetime.datetime.combine(date2, datetime.time(0, 0)):
                    logouts.append(logout)
                if total_hours == 0:
                    total_hours = (logout - startdate)
                else:
                    total_hours += (logout - startdate)
    st.write(f'Total Hours: {total_hours}')

st.subheader('Create New Employee')
new_employee=st.text_input('New Employee Name')
if st.button('Add New Employee'):
    cur.execute('INSERT INTO employees(name) VALUES(?)',(str(new_employee),))
    con.commit()
    st.write(f'Added {new_employee}!')
    st.experimental_rerun()

```



9.15 Administrador de archivos FTP

Esta aplicación permite a los usuarios descargar y cargar archivos desde/hacia cuentas de servidores FTP, utilizando la información de inicio de sesión guardada para cada servidor. Los usuarios eligen de una lista de servidores predefinidos para iniciar sesión y eligen los archivos seleccionados para descargar. Los archivos seleccionados de un disco duro local también se pueden cargar en un servidor elegido:

```

import streamlit as st, ftplib, os

servers = {
    'account1' : ['myaccounturl1.com', 'username1', 'password1', '/myfolder'],
    'account2' : ['myaccounturl2.com', 'username2', 'password2', '/public_html'],
    'account3' : ['myaccounturl3.com', 'username3', 'password3', '/afolder']
}

if 'files' not in st.session_state:

```



```

    st.session_state.files=[]

def list_ftp_files(FTP_HOST, FTP_USER, FTP_PASS, FTP_FOLDER):
    ftp = ftplib.FTP(FTP_HOST, FTP_USER, FTP_PASS)
    ftp.encoding = "utf-8"
    ftp.cwd(FTP_FOLDER)
    files = ftp.nlst()
    ftp.quit()
    return files

def download_ftp_file(FTP_HOST, FTP_USER, FTP_PASS, FTP_FOLDER, FTP_FILENAME):
    ftp = ftplib.FTP(FTP_HOST)
    ftp.login(FTP_USER, FTP_PASS)
    ftp.cwd(FTP_FOLDER)
    try:
        ftp.retrbinary("RETR " + FTP_FILENAME , open(FTP_FILENAME, 'wb').write)
    except:
        st.write("Error downloading")
    with open(FTP_FILENAME, 'r+') as file:
        return file.read()

def upload_ftp_file(
    FTP_HOST,
    FTP_USER,
    FTP_PASS,
    FTP_FOLDER,
    FTP_FILENAME,
    file
):
    ftp = ftplib.FTP(FTP_HOST)
    ftp.login(FTP_USER, FTP_PASS)
    ftp.cwd(FTP_FOLDER)
    with open(FTP_FILENAME, "wb") as binary_file:
        binary_file.write(file)
    with open(FTP_FILENAME, 'rb') as myfile:
        try:
            ftp.storbinary("STOR " + FTP_FILENAME, myfile)
        except:
            st.write('Error uploading')

serverlist=[key for key in servers.keys()]
selected_server=st.selectbox('Select a Server', serverlist)
if selected_server:
    if selected_server!=' ':
        login = servers[selected_server]
        FTP_HOST = login[0]
        FTP_USER = login[1]
        FTP_PASS = login[2]
        FTP_FOLDER = login[3]
        st.session_state.files = list_ftp_files(
            FTP_HOST,
            FTP_USER,
            FTP_PASS,
            FTP_FOLDER
        )
        for i in ['.', '..']:
            st.session_state.files.remove(i)
            st.session_state.files.insert(0, ' ')

uploaded_file = st.file_uploader('Upload File')
if uploaded_file:
    login = servers[selected_server]
    FTP_HOST = login[0]
    st.write('Upload Complete, Close File Name to Update Selection List')
    FTP_USER = login[1]
    FTP_PASS = login[2]
    FTP_FOLDER = login[3]
    FTP_FILENAME = uploaded_file.name
    file = upload_ftp_file(
        FTP_HOST,
        FTP_USER,
        FTP_PASS,
        FTP_FOLDER,
        FTP_FILENAME,
        uploaded_file.getvalue()
    )

```

```

selected_file=st.selectbox('Select File to Download', st.session_state.files)
if selected_file and selected_file!=' ':
    login = servers[selected_server]
    FTP_HOST = login[0]
    FTP_USER = login[1]
    FTP_PASS = login[2]
    FTP_FOLDER = login[3]
    FTP_FILENAME = selected_file
    file = download_ftp_file(
        FTP_HOST,
        FTP_USER,
        FTP_PASS,
        FTP_FOLDER,
        FTP_FILENAME
    )
    st.download_button('Download', file, file_name=selected_file)

```



10. Instalación y ejecución de Streamlit en una computadora local

Hasta ahora, hemos utilizado el conveniente editor en línea de livecode.streamlitapp.com para explorar las funciones de Streamlit.

Para uso en producción, Streamlit tarda solo unos minutos en configurarse localmente en la mayoría de las computadoras modernas, y se puede autohospedar utilizando cualquier proveedor de servicios en la nube conocido, con o sin contenedores Docker (o puede alojar aplicaciones de forma gratuita en el servicio en la nube de Streamlit).

Para comenzar a usar Streamlit en su propia PC local, necesitará una versión de Python de 64 bits de python.org instalada, que se ejecute en un sistema operativo de 64 bits. Cuando tenga Python instalado, ejecute 'pip install streamlit' en la línea de comandos de su sistema operativo (ejecute 'cmd' en Windows o la Terminal en una Mac para acceder a la línea de comandos):



Una vez completado el proceso de instalación de pip, puedes comprobar que Streamlit se está ejecutando ejecutando 'streamlit hello' en el símbolo del sistema. Streamlit debería abrir el navegador de forma predeterminada en <http://localhost:8501>, donde inicializará la aplicación de demostración ejecutando:



Para crear su primera aplicación Streamlit, puede usar cualquier editor de texto para crear un nuevo archivo de texto. El uso de un IDE como Microsoft Visual Studio Code (sin <https://code.visualstudio.com/Download>) puede proporcionar un útil autocompletado de código, además de funciones útiles de gestión de proyectos y ejecución integrada, pero incluso el Bloc de notas se puede usar para comenzar a editar aplicaciones Streamlit. Comience guardando el siguiente código en un nuevo archivo llamado 'streamlit1.py':

```

import streamlit as st
if st.button('Say Hello'):
    st.write('Hello World!')

```

En la línea de comandos, vaya a la carpeta donde ha guardado streamlit1.py. Por ejemplo, si ha guardado streamlit1.py en su carpeta Documentos, escriba 'documentos de CD' (si nunca ha usado el comando 'CD' en la consola de su computadora, puede encontrar muchos tutoriales con una búsqueda rápida en Google):

Puede escribir 'dir' en la línea de comandos para mostrar una lista de los archivos en la carpeta actual, para asegurarse de que está en la carpeta correcta. Ahora escriba 'streamlit run streamlit1.py' y se abrirá su navegador, mostrando la aplicación web creada por el código anterior. Haga clic en el botón 'Decir hola' y la aplicación lo recibirá con el mensaje '¡Hola mundo!'.

De vuelta en la ventana de la consola, verá que se proporcionan 2 direcciones para la aplicación. La primera dirección es la 'http://localhost:8501', que es la dirección que puede usar para ver la aplicación, en la máquina donde se ejecuta Streamlit. La segunda dirección contiene la dirección IP del equipo en el que se ejecuta la aplicación Streamlit. Puede escribir esa URL en la barra de direcciones en cualquier navegador moderno en cualquier computadora o teléfono móvil / tableta conectado a su red WIFI de área local, y verá que la aplicación se ejecuta en ese dispositivo. Streamlit ajusta automáticamente los diseños de forma receptiva, para que se muestren correctamente en máquinas móviles y de escritorio sin ningún cambio de código. Si tienes un dispositivo móvil u otro ordenador conectado a tu red WIFI, prueba tu aplicación. Cuando haya terminado, puede cerrar el servidor de

aplicaciones Streamlit enfocando la ventana de línea de comandos y presionando las teclas CTRL + C en su teclado.



A continuación, intente ejecutar cualquiera de los ejemplos de base de datos CRUD o un ejemplo de aplicación de API de anteriormente en el tutorial. Simplemente escriba 'streamlit run ' para ejecutar el código Streamlit, usando cualquier nombre de archivo que cree.

Streamlit se puede usar así, ejecutándose en una red de área local, para usuarios que prefieren ejecutar aplicaciones internas con archivos confidenciales almacenados solo en máquinas internas. La ejecución de una instalación local también proporciona una configuración sencilla para el ciclo de desarrollo, antes de publicar aplicaciones en línea.

Cabe señalar que cada vez que cambie el archivo fuente de una aplicación Streamlit, la aplicación detectará el cambio en el código fuente y preguntará al usuario si la aplicación debe volver a ejecutarse con el nuevo código. Esto hace que los ciclos de depuración y creatividad se muevan mucho más rápido, sin necesidad de herramientas externas además de Streamlit. Simplemente edite su código Streamlit, mantenga su aplicación ejecutándose en un navegador, seleccione 'volver a ejecutar siempre' en la aplicación y la aplicación en ejecución se actualizará automáticamente cada vez que guarde cambios en el código:



11. Sitios optimizados de varias páginas

Hay varias formas de crear aplicaciones Streamlit con múltiples pantallas de información.

11.1 Uso de contenedores y St.Sidebar()

Cualquier widget de Streamlit se puede colocar en un diseño de barra lateral usando la sintaxis: st.sidebar. (). Esto es útil para crear fácilmente varios diseños de pantalla dentro de un solo script simple:

```
import streamlit as st, numpy as np, pandas as pd

menu=st.sidebar.selectbox('Menu', ['Page 1', 'Page 2', 'Page 3'])
if menu=='Page 1':
    st.header('Page 1')
    df=pd.DataFrame(np.random.randn(10,5),columns=('c %d'% i for i in range(5)))
    st.table(df)
if menu=='Page 2':
    st.header('Page 2')
    st.json({'foo':'bar','baz':'boz','stuff':['stuff 1','stuff 2']})
if menu=='Page 3':
    st.header('Page 3')
```



11.2 Verdaderas aplicaciones de varias páginas a partir de múltiples scripts

Para crear sitios reales de varias páginas compuestos por varios scripts, consulte:

<https://blog.streamlit.io/introducing-multipage-apps/>

<https://docs.streamlit.io/library/get-started/multipage-apps>

12. Uso de FastAPI para servir datos manipulados por aplicaciones Streamlit

Las aplicaciones web modernas suelen comunicarse a través de llamadas API HTTP (Rest). Puede usar una biblioteca como FastAPI para leer y escribir datos en la misma base de datos a la que se conectan sus aplicaciones Streamlit. Esta funcionalidad permite que las aplicaciones creadas en cualquier lenguaje de programación interactúen con sus aplicaciones Streamlit.

Aquí hay un ejemplo simple que demuestra cómo usar FastAPI para servir datos que han sido creados por el ejemplo de base de datos Streamlit en la sección anterior del tutorial. Simplemente colóquelo en la misma carpeta que el código Streamlit:

```

import sqlite3
con=sqlite3.connect('db.db')
cur=con.cursor()

import uvicorn
from fastapi import FastAPI
app = FastAPI()

@app.get("/{col}")
async def get_col(col: str):
    return [row for row in cur.execute(f"SELECT {col} FROM db;")]

if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8000)

```

Vaya a http://localhost:8000/* en su navegador y verá todas las filas de la base de datos, devueltas por la API en formato json. Vaya a <http://localhost:8000/name>, <http://localhost:8000/letters> o <http://localhost:8000/note> y verá las columnas seleccionadas de la base de datos devueltas. El json generado por este código FastAPI puede ser consumido por cualquier marco web, para permitir que las aplicaciones escritas en cualquier plataforma interoperen y compartan datos fácilmente.

El siguiente ejemplo incorpora tanto el código Streamlit como el código FastAPI en un solo script, y permite al usuario ejecutar el servidor FastAPI directamente desde la interfaz Streamlit (actualmente configurada para el sistema operativo Windows). Esta es realmente una solución de pila completa, con interfaz de usuario, base de datos e interfaz API, en menos de 50 líneas de código:

```

import streamlit as st, sqlite3, ast
con=sqlite3.connect('db.db')
cur=con.cursor()
cur.execute('CREATE TABLE IF NOT EXISTS db(name TEXT, letters TEXT, note TEXT)')

if st.button('Add New Row'):
    cur.execute('INSERT INTO db(name, letters, note) VALUES(?,?,?)', ('','[]',''))
    con.commit()

for row in cur.execute('SELECT rowid, name, letters, note FROM db ORDER BY name'):
    with st.expander(row[1]):
        with st.form(f'ID-{row[0]}', clear_on_submit=True):
            name=st.text_input('Name', row[1])
            letters=st.multiselect('letters', ['A', 'B', 'C'], ast.literal_eval(row[2]))
            note=st.text_area('Note', row[3])
            if st.form_submit_button('Save'):
                cur.execute(
                    'UPDATE db SET name=?, letters=?, note=? WHERE name=?;',
                    (name, str(letters), note, str(row[1]))
                )
                con.commit()
                st.experimental_rerun()
            if st.form_submit_button("Delete"):
                cur.execute(f'DELETE FROM db WHERE rowid="{row[0]}"')
                con.commit()
                st.experimental_rerun()

apicode=''import sqlite3
con=sqlite3.connect('db.db')
cur=con.cursor()
import uvicorn
from fastapi import FastAPI
app = FastAPI()
@app.get("/{col}")
async def get_col(col: str):
    return [row for row in cur.execute(f"SELECT {col} FROM db;")]
if __name__ == '__main__':
    uvicorn.run(app, host='0.0.0.0', port=8000)'''

if st.button('Run Web API'):
    with open("api.py", "w") as file: file.write(apicode)
    import os ; os.system('start cmd /K python api.py')
    st.markdown(f"""[View API] (http://localhost:8000/\*) If the API
    doesn't start, run 'python api.py' in your OS console & browse
    localhost:8000/*""", unsafe_allow_html=True)

```

13. Configuración de la página

Streamlit le permite establecer algunos parámetros de diseño de página con `st.set_page_config()`:

```
st.set_page_config(page_title="Sqlite Images", layout="wide")
```

`St.set_page_config()` debe ser el primer método Streamlit utilizado en una aplicación y solo se puede configurar una vez en cada aplicación. Tenga en cuenta que `st.set_page_config()` *no se ejecutará en el editor de livecode* utilizado en este tutorial porque ese método ya está incluido en el código del editor.

Para realizar otros ajustes de diseño en tu app, puedes incluir estilos CSS en un widget `st.markdown()`, con el argumento `'unsafe_allow_html'` establecido en `True`. El siguiente CSS oculta el enlace predeterminado 'Made With Streamlit' (en la parte inferior de la página) y el menú de hamburguesas (esquina superior derecha) en las aplicaciones, y elimina el espacio vacío predeterminado en la parte superior de las aplicaciones. Esto puede ser útil cuando se producen aplicaciones de cara al público:

```
st.markdown(
    '''<style>
        #MainMenu{visibility: hidden;} footer{visibility: hidden;}
        #root>div:nth-child(1)>div>div>div>div>section>div{padding-top: .2rem;
    </style>''', unsafe_allow_html=True
)
```



Aquí hay una versión más grande del ejemplo anterior de CRUD de forma única, donde el diseño más amplio es útil:

```
import streamlit as st, pandas as pd, sqlite3, ast

# st.set_page_config(layout="wide", page_title="Sqlite CRUD")
# uncomment the line above if running outside the livecode editor

st.markdown(
    '''<style>
        #MainMenu{visibility: hidden;} footer{visibility: hidden;}
        #root>div:nth-child(1)>div>div>div>div>section>div{padding-top: .2rem;
    </style>''', unsafe_allow_html=True
)

con=sqlite3.connect('crud.db')
cur=con.cursor()
cur.execute(
    '''
        CREATE TABLE IF NOT EXISTS crud (
            namecol TEXT,
            passwordcol TEXT,
            agecol TEXT,
            giftcol TEXT,
            colorcol TEXT,
            lettercol TEXT,
            notescol TEXT
        )
        # key_col INTEGER NOT NULL PRIMARY KEY,
    '''
)

st.title('CRUD Example')
col1, col2=st.columns([2,3])
with col1:
    st.header("Create/Update/Delete")
    names=[
        str(row[0]) for row in cur.execute(
            f"SELECT namecol FROM crud;"
        )
    ]
    names.insert(0, '')
    name_to_update=st.selectbox('', names)
```



```

        gift,
        color,
        str(letters),
        notes,
        str(nameval)
    )
)
con.commit() # con.close()
st.experimental_rerun()
if st.form_submit_button("Delete"):
    cur.execute(
        f'''DELETE FROM crud WHERE namecol="{name_to_update}";'''
    )
    con.commit()
    st.write(f"{name_to_update} has been deleted")

with col2:
    st.header("Read")
    if st.checkbox('Recheck to refresh display', value=True):
        myrows=[
            list(row) for row in cur.execute(
                '''
                SELECT
                    rowid,
                    namecol,
                    passwordcol,
                    agecol,
                    giftcol,
                    colorcol,
                    lettercol,
                    notescol
                FROM crud
                ORDER BY namecol
                '''
            )
        ]
    st.dataframe(
        pd.DataFrame(
            myrows,
            columns=[
                'rowid',
                'name',
                'password',
                'age',
                'gift',
                'color',
                'letters',
                'notes'
            ]
        ),
        height=800
    )

```



14. A dónde ir desde aquí

14.1 Documentación, blog y foro de Streamlit

La documentación de la API y el código de ejemplo para todos los widgets y funciones disponibles en Streamlit, así como los tutoriales y la información sobre el alojamiento con la nube de Streamlit, están disponibles en:

<https://docs.streamlit.io/>

El Blog es una excelente manera de mantenerse al día con las nuevas funciones (¡Streamlit está evolucionando rápidamente!):

<https://blog.streamlit.io/>

El foro de la comunidad de Streamlit es un gran lugar para buscar respuestas y preguntas:

<https://streamlit.io/components>

14.2 Componentes Streamlit

La comunidad de Streamlit ha creado algunos componentes de terceros extremadamente poderosos. El componente `aggrid` es un puerto de Streamlit de la conocida cuadrícula de JavaScript, que proporciona muchas características que no están disponibles en el widget nativo `st.dataframe()` de Streamlit:

<https://pablofonseca-streamlit-aggrid-examples-example-jyosi3.streamlitapp.com/>

El componente WebRTC es útil si desea realizar cualquier tipo de procesamiento de video. Incluso hay una aplicación de videoconferencia con todas las funciones construida con él en solo unas pocas líneas de código:

<https://webrtc.streamlitapp.com/>

El componente Autenticación de usuario es útil cuando desea proporcionar funciones de inicio de sesión de nombre de usuario y contraseña a una aplicación.

[Tutorial de autenticación de ToWardsDatascience](#)

Y hay muchos más:

<https://docs.streamlit.io/components>

15. Sobre el autor

Nick Antonaccio ha estado escribiendo código de software desde 1979. Busque en Google 'aprender rebol', 'aprender código en vivo', 'aprender ns basic', 'aprender python anvil', 'jslinb', 'aprender rfo', 'aprender haxe', 'aprender programación roja', 'tutorial de API de etsy', 'programación empresarial', y verá sitios de Nick entre los primeros resultados en cada categoría. Nick ha escrito software de misión crítica para grandes corporaciones y cientos de pequeñas empresas. Su software de consignación Merchant's Village se ha utilizado para vender decenas de millones de productos en tiendas minoristas físicas (busque 'consignación de código abierto': ha sido el primer resultado en Google durante más de una década).

A Nick le apasiona ayudar a otros propietarios de negocios a mejorar sus condiciones operativas y resultados finales con tecnología. Para obtener más información, consulta:

<https://merchantsvillage.com/About%20The%20Author.html>

Para contratar a Nick para tutoría, o si desea desarrollar un producto de software, llame / envíe un mensaje de texto al 215-630-6759 o envíe un mensaje a nick@com-pute.com

